Universidad Carlos III de Madrid
Escuela Politécnica Superior

Desarrollo de una aplicación de proyección distribuida para dispositivos móviles basada en tecnologías inalámbricas de comunicaciones.

Proyecto Fin de Carrera
Ingeniería Técnica de Telecomunicaciones: Telemática

Autor: Víctor Parrilla Martín
Tutor: Dr. Mario Muñoz Organero
Junio 2009

PROYECTO FIN DE CARRERA

Departamento de Ingeniería Telemática
Universidad Carlos III de Madrid

Título: Desarrollo de una aplicación de proyección distribuida para dispositivos móviles basada en tecnologías inalámbricas de comunicaciones.

Autor: Víctor Parrilla Martín

Tutor: Dr. Mario Muñoz Organero

EL TRIBUNAL

Presidente:

Secretario:

Vocal:

Realizado el acto de defensa del Proyecto Fin de Carrera el día 29 de Junio de 2009 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de:

Presidente Secretario Vocal

“Sometimes the only path to take is the hardest one to walk…” - Jim Hesketh

Agradecimientos.

A mis padres, Candido y Rosa, sencillamente por ser mis dos mayores pilares en los que me he apoyado, y ya que gracias a ellos puedo estar orgulloso de lo que soy hoy en día.

A mi hermana Rosa, por todas las cosas que le debo y que soy incapaz de resumir en estas líneas. Ha sido – y espero que lo siga siendo - una fuente de inspiración en mucho momentos de mi vida. Quien diría cuando éramos pequeños y nos peleábamos en casa día si y día también que hoy en día nos llevaríamos tan bien.

A todos mis compañeros de universidad, tanto los que he tenido durante mi corto paso por la superior de Teleco, como a los recientes en Telemática, ya que desde el primer día han hecho que el paso por la universidad se haya hecho mas llevadero y ameno.

A mi novia Sara, por estar a mi lado siempre que lo he necesitado, tanto en los buenos como en los malos momentos. *"You're the piece that fits in my life"*

A mi tutor del proyecto, Mario, por su apoyo y paciencia en el transcurso del desarrollo de mi proyecto fin de carrera.

Resumen.

En la actualidad las comunicaciones inalámbricas están tomando una gran importancia y se encuentran en pleno auge, todo ello motivado principalmente por el interés de querer comunicar e interconectar dispositivos sin necesidad de cables. Las emergentes tecnologías inalámbricas han facilitado la conexión entre dispositivos que no se encuentran juntos físicamente o que resulta costoso el cableado entre ellos.

Es probable que en un futuro no podamos imaginarnos como hemos podido vivir sin estas tecnologías sin cables. Aunque esto no quiere decir que, al menos a corto plazo, las comunicaciones inalámbricas vayan a sustituir a las comunicaciones tradicionales cableadas, sino que serán un complemento y podrán coexistir en armonía perfectamente.

Las capacidades que ofrece la tecnología inalámbrica proporcionan mayor comodidad y movilidad con total funcionalidad en cualquier lugar. Pero para que tenga aceptación entre los usuarios, esta funcionalidad debe garantizarse cualquiera que sea la plataforma o la marca que adquieran. Y para ello, los posibles competidores en este mercado se están poniendo de acuerdo para establecer estándares que aseguren a los usuarios finales la compatibilidad y/o el funcionamiento conjunto de sus distintos productos.

En este escenario de comunicaciones inalámbricas, podemos ver a los dispositivos móviles como elementos importantes e imprescindibles, que a pesar de llevar ya unos cuantos años entre nosotros, siguen en pleno auge en la actualidad. Los móviles se han integrado de manera casi instantánea en nuestras vidas, casi sin haber sido conscientes de ellos, pero que hoy en día no podríamos estar sin ellos. Día a día salen teléfonos nuevos, que incorporan nuevas funcionalidades o mejoras con respecto a sus anteriores versiones, evolucionando constantemente.

Este proyecto trata de acercarnos a las tecnologías inalámbricas, en concreto Bluetooth y Near Field Communication (NFC), integradas en dispositivos móviles. Para ello se desarrollara una herramienta en un entorno distribuido donde varios dispositivos móviles haciendo uso de ambas tecnologías podrán proyectar archivos almacenados en éstos.

Abstract.

Wireless Communications are beginning to gain great importance throughout these days. This development is due to the fact that many people wish to communicate and connect devices without any type of cable connection. The uprising wireless communications make our job easier whenever we try to connect devices which are clearly and physicably separated or just because cable connections are very expensive endeed.

In a near future people will trivialize our past experience with cable devices. This does not exactly mean that one technology will substitute the other, in fact, wireless communications will become a complementary service, living in harmony with the old-fashioned technology.

Wireless technology is remarkably handy and able to move around and work elsewhere. For this development to be accepted between its users, any brandname or launchpad has to guarantee its functionality. The market and its competitors have to come to an agreement in order to stablish the same standards in compatibility and operations of its products; all for the public.

In this scenery we can see how mobile telephones take the lead. Although they have joined us in a fairly recent time, they actually have this increasing popularity which make them easily fittable in our society. We could not live without them and seldom the days go by, new phones begin to appear. This new devices incorporate many different features, leaving behind its old versions.

This proyect tries to bring up close this wireless technologies, Bluetooth and Near Fiel Communicaction (NFC), which are clearly integrated in mobile devices. A new tool will be developed in a distributed environment were several mobile devices will use both technologies in order to screen archives stored in them.

INDICE

Capitulo 1.

Introducción.

En este primer capítulo introductorio se incluye una breve descripción de la historia de las tecnologías inalámbricas con el fin de introducir el marco de trabajo y las motivaciones de este proyecto. Se hace una breve descripción de los distintos objetivos a alcanzar, así como una descomposición del proyecto en fases de trabajo y una breve descripción de la estructura de la memoria.

1.1 Breve historia de las tecnologías inalámbricas.

Desde los inicios de la humanidad, un tema fundamental con respecto al desarrollo y el progreso, ha sido la necesidad de comunicación entre unos y otros, presente a lo largo de la historia. En los últimos años los nuevos logros de la tecnología han sido la aparición de ordenadores, líneas telefónicas, dispositivos móviles, redes inalámbricas, etc., permitiendo un gran abanico de posibilidades en el entorno de las comunicaciones.

La aplicación de la tecnología inalámbrica, viene teniendo un gran auge en velocidades de transmisión, aunque sin competir con la utilización de redes cableadas o el uso de la fibra óptica, sin embargo cubren satisfactoriamente la necesidad del movimiento de los usuarios.

Este tipo de comunicaciones sin cable esta evolucionando de manera espectacular, y cada día se consiguen mejorar un gran numero de aspectos de tal manera que llegará un punto en el que las tecnologías inalámbricas no tengan nada que envidiar a las tecnologías cableadas.

Dentro de las tecnologías inalámbricas se podrían nombrar muchas, pero este proyecto se centrará en dos, Bluetooth y Near Field Communication, que a día de hoy se encuentran en pleno auge y cada vez más extendidas y popularizadas entre los usuarios.

1.2 Motivación.

La principal motivación de este proyecto es la constante evolución y la importancia que están adquiriendo las tecnologías inalámbricas hoy en día. Si a eso le sumamos la integración que están teniendo los dispositivos móviles, es notable destacar las enormes posibilidades que eso nos ofrece.

1.3 Objetivos del proyecto.

Varios son los objetivos de este proyecto. Se enumeran a continuación:

1. Analizar y estudiar las tecnologías inalámbricas, y mas en concreto, Bluetooth y NFC, desde los principales aspectos técnicos de funcionamiento, escenarios de uso, historia, etc.

2. Estudiar por qué estas tecnologías ofrecen un gran número de posibilidades, no solo en la integración con los dispositivos móviles, sino en con la combinación con otra tecnologías actuales.

3. Estudiar las ventajas que ofrecen estas tecnologías en un entorno de empresa, y más en concreto en un escenario de reuniones.

4. Implementar y desarrollar una herramienta que permita la proyección de manera distribuida de contenidos multimedia, a través del uso de un teléfono móvil.

5. Desarrollar una batería de pruebas que permita testear y comprobar el correcto funcionamiento de esta herramienta.

6. Estudiar las posibles líneas de futuro que podrían desarrollarse para mejorar ésta herramienta.

1.4 Fases del proyecto.

El desarrollo del presente proyecto se ha descompuesto, desde su inicio hasta su conclusión, en varias fases distintas cubriendo cada una de ellas diferentes aspectos del desarrollo. Podemos descomponer el proyecto en seis fases que enumeramos a continuación:

- Fase 0: documentación y configuración del sistema.
- Fase 1: desarrollo de la funcionalidad NFC.
- Fase 2: desarrollo de la funcionalidad Bluetooth.
- Fase 3: desarrollo de la funcionalidad File-browsing.
- Fase 4: pruebas y depuración de la aplicación.
- Fase 5: redacción de la memoria y documentación del proyecto.
- Fase 6: corrección.

1.5 Contenido de la memoria.

El siguiente capitulo introduce el estado del arte, analizando las tecnologías en las que estará basado este proyecto, como son Bluetooth y NFC. Además se analizará el entorno y plataforma J2ME para el desarrollo de dispositivos móviles como teléfonos móviles y PDAs. Y por ultimo se analizarán los dispositivos móviles, su historia, las diferentes generaciones y se verán las características del teléfono móvil que se usará en este proyecto (Nokia 6131 NFC).

En el tercer capitulo se presentarán los objetivos del proyecto, así como todo el desarrollo de la herramienta, desde su diseño inicial, hasta las pruebas y depuración, pasando por toda la implantación de cada una de las funcionalidades de las que consta.

En el capítulo cuarto se presentan las conclusiones del proyecto así como una serie de futuras líneas de trabajo que podrían seguirse en caso de que alguien estuviera interesado en proseguir con este proyecto.

El quinto capitulo presenta el desarrollo temporal del proyecto, así como un pequeño presupuesto del mismo.

Por último, además de todas las referencias bibliográficas que se han usado para este proyecto, se incluyen dos anexos, que no son más que un manual de instalación de las aplicaciones necesarias para el desarrollador/programador y un manual de usuario para poder arrancar servidor y cliente.

Capítulo 2.

Estado del arte.

En este capitulo se hace una breve introducción a las tecnologías utilizadas en este proyecto. En primer lugar se presentarán de las tecnologías inalámbricas, una muy conocida hoy en día como es Bluetooth y otra menos extendida como es NFC. Sobre ambas podremos conocer como funcionan, algunas características tales como banda de frecuencia sobre la que operan o que tasa de bits alcanzan en las comunicaciones, usos o campos de aplicación, etc.

En segundo lugar se presentan los dispositivos móviles, muy presentes hoy en día y prácticamente sin los que no podríamos vivir al ser casi parte imprescindible en nuestras vidas.

Por ultimo se presentara el lenguaje Java, y mas en concreto la plataforma J2ME, clave esencial para el desarrollo de este proyecto.

2.1 Near Field Communication - NFC

2.1.1 Introducción.

Entre las tecnologías de corto alcance o proximidad, ha surgido un gran interés en los últimos meses por la tecnología Near Field Communication (NFC). Esta tecnología comenzó a desarrollarse en el año 2002 en una acción conjunta de Philips y Sony, con el fin de conseguir un protocolo compatible con las tecnologías contactless propietarias existentes en el mercado, FeliCaTM (Sony) y MifareTM (Philips).

La tecnología NFC es una tecnología de interconexión de dispositivos que opera a la frecuencia de 13,56 MHz (banda en la que no se necesita licencia administrativa), y a una distancia de pocos centímetros (de 0 a 20 cm., típicamente 10 cm.). Surge de la evolución de la tecnología de identificación de dispositivos por radiofrecuencia (RFID, Radio Frecuency Identification) incluyendo funciones de las tecnologías de interconexión de redes y de tarjetas inteligentes.

2.1.2 Un poco de historia, RFID.

2.1.2.1 ¿Que es RFID?

RFID responde a las siglas de Radio-Frequency IDentification. Es un método automático de identificación, que permite extraer y guardar datos remotamente usando los dispositivos conocidos como etiquetas (o transpondedores).

Una etiqueta de RFID es un objeto el cual puede ser aplicado o incorporado en un producto, un animal, o una persona con el fin de la identificación usando ondas de radio. Algunas etiquetas se pueden leer a muchos metros del lector.

La mayoría de las etiquetas de RFID contienen por lo menos dos partes. Uno es un circuito integrado que permite salvar y procesar la información, modular y demodular una señal de radiofrecuencia (RF), y otras funciones especializadas. El segundo es una antena para recibir y transmitir la señal.

Hay otro tipo de tarjetas, Chipless RFID, que permiten la identificación sin circuito integrado, de tal modo que se puedan imprimir directamente sobre etiquetas, lo que supone un coste menor que las otras etiquetas tradicionales.

2.1.2.2 Campos de aplicación.

Hoy en día, el uso de RFID está llegando a ser cada vez más frecuente, en gran medida debido a la reducción del precio de esta tecnología, lo que le esta permitiendo poder ser aplicada en una gran variedad de campos.

Medición en carreras.

La tecnología RFID ha sido incorporada a esta materia muy recientemente. Gracias a ello, se pueden registrar los tiempos de numerosos participantes en carreras donde la participación es muy numerosa o donde es muy tediosa o difícil la medida de estos tiempos para cada uno de ellos.

Cada individuo lleva un dorsal con su número de participante y además una etiqueta RFID integrada que puede ser leída por la antena colocada en la meta (o en las zonas de paso). Los errores y accidentes en las mediciones se evitan puesto que cualquier persona puede comenzar y acabar en cualquier momento sin tener que ser tratado en modo de lotes.

Este método está siendo adaptado por muchas agencias del reclutamiento que tengan un PET (prueba de resistencia física) como procedimiento calificativo especialmente en caso de que el volumen de candidatos sea muy grande.

Pasaportes.

Las etiquetas de RFID se están utilizando en los pasaportes expedidos por muchos países, como Malasia (principios de 2000), Nueva Zelanda (2005), Bélgica, los Países Bajos (2005), Noruega (2005), Irlanda (2006), Japón (2006), Paquistán, Alemania, Portugal, Polonia (2006), el Reino Unido, Australia y los Estados Unidos (2007).

Los primeros pasaportes de RFID (“e-pasaportes”) fueron publicados en Malasia. Además de la información personal contenida en las paginas del pasaporte, éstos incluyen un registro de entradas y salidas del país (hora, fecha, y lugar).

Inicialmente se establecía que las etiquetas podrían leerse solamente a una distancia de 10 centímetros, pero más tarde se vio que pueden leerse los pasaportes hasta una distancia de 10 metros.

Los pasaportes fueron diseñados para incorporar una fina lámina de metal que evitara que programas de lectura desautorizados accedan a la información cuando el pasaporte es cerrado. Otro método de seguridad era que antes de que la etiqueta de un pasaporte pudiera ser leída, ésta debía ser registrada en un programa de lectura de RFID.

Algunos otros países europeos están planeando agregar huellas digitales y otros datos biométricos, mientras que algunos ya lo han hecho.

Pago en transportes.

Hoy en día, en un gran numero de países, tanto de Europa, Asia y América, han incorporado esta tecnología en el sector de los transportes. Uno de estos usos es el sistema de pago en peajes a través de etiquetas RFID esta implantado. En muchas

ciudades esta implantado el pago en el transporte publico con éstas tarjetas (Metrorail en USA, SUICa en Japón, etc.).

Estos nuevos usos permiten automatizar tareas, ahorrar tiempo en el cobro a los usuarios, además de suponer un ahorro de personal en algunos otros casos.

Seguimiento y control de mercancías.

En este campo, podemos ver como las mercancías han comenzado a incluir etiquetas RFID, lo que permite una lectura rápida en controles o a la entrada de grandes almacenes o mercados. Esto agiliza mucho el proceso de conteo de mercancías, pues con el método tradicional era necesario hacer un inventario. Pero con RFID el control de la mercancía es tan sencillo como que el camión que la transporta pase por un debajo del lector de radiofrecuencia situado en las puertas del almacén.

Identificación animal.

Muy conocido es el uso de los chips en animales domésticos. Hoy en día es muy común que nuestras mascotas, como por ejemplo perros y gatos, posean un microchip incorporado con información personal acerca del animal. Esto es muy útil cuando un usuario va con su mascota a un veterinario pues se puede obtener de manera muy rápida una gran cantidad de información acerca del animal. También es de gran utilidad en el caso de animales extraviados o perdido, pues basta con leer su chip para axial poder localizar al dueño.

Implantes humanos.

Esta es una técnica no muy utilizada a día de hoy, pero que quizás en un futuro se empiece a emplear en numerosas aplicaciones. El implante consiste en un chip similar al que se le pone a las mascotas y contiene información del portador. Algunas discotecas o clubes de Barcelona y Rótterdam han permitido el uso de microchips en sus clientes VIP que lo emplean para el pago automático de bebidas.

Una posible aplicación sería la identificación de personas, aunque expertos en seguridad desaconsejan este uso debido a la relativa facilidad en que un usuario podría robarle la identidad a otro mediante un ataque de “man-in-the-middle”.

Tiendas y grandes superficies.

Este uso lleva mucho tiempo extendido, sustituyendo al tradicional código de barras por etiquetas RFID, en productos como libros, cd's o dvd's. La incorporación de estas etiquetas no solo permite el acceso a la información del producto mediante la lectura de su etiqueta si no que también actúa como elemento de seguridad.

Otros usos.

Se está empezando a usar RFID, en museos, hospitales, zoológicos, campos de golf, casinos, etc. Por lo que se puede ver, RFID es una tecnología que puede aplicarse a un gran número de campos cada vez mayor.

2.1.3 Principales características de NFC.

- NFC es una solución desarrollada por Nokia, Sony y Philips y esta basada en RFID + tecnologías interconectadas.
- Trabaja en la banda de los 13,56 Mhz, por lo que no se le aplica ninguna restricción y no requiere ninguna licencia para su uso.
- Las velocidades de transmisión soportadas actualmente son de 106, 212 y 424 Kbps.
- Se puede usar para configurar e iniciar otras conexiones wireless como son Bluetooth, Wi-fi o UltraWireband.
- Cuando se enciende el lector, emite una señal de radio de corto alcance que activa el microchip de la etiqueta con lo que podremos leer una pequeña cantidad de datos que se encuentra almacenado en ella.
- Tecnología inherentemente segura, ya que por su corto alcance los dispositivos tienen prácticamente que tocarse.
- Proporciona un modo de acceso a los servicios muy familiar e intuitivo a los usuarios: “si quieres un servicio, tócalo”. Por ejemplo, si quieres comprar un refresco con el terminal, “toca la máquina”.
- No precisa configuración por parte del usuario.

2.1.4 Modos de funcionamiento.

Pasivo: Sólo un dispositivo genera el campo electromagnético y el otro se aprovecha de la modulación de la carga para poder transferir los datos. El iniciador de la comunicación es el encargado de generar el campo electromagnético.

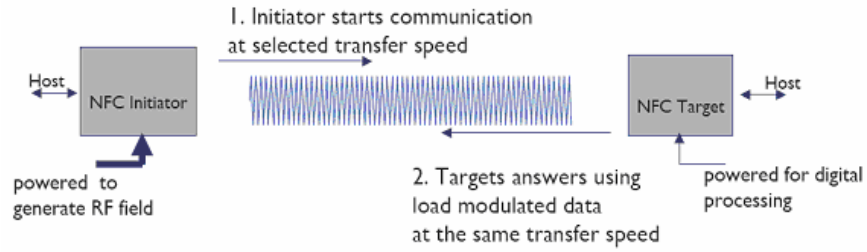

Ilustración 1. Modo pasivo de funcionamiento.

Activo: Ambos dispositivos generan su propio campo electromagnético, que utilizarán para transmitir sus datos. Ambos dispositivos necesitan energía para funcionar.

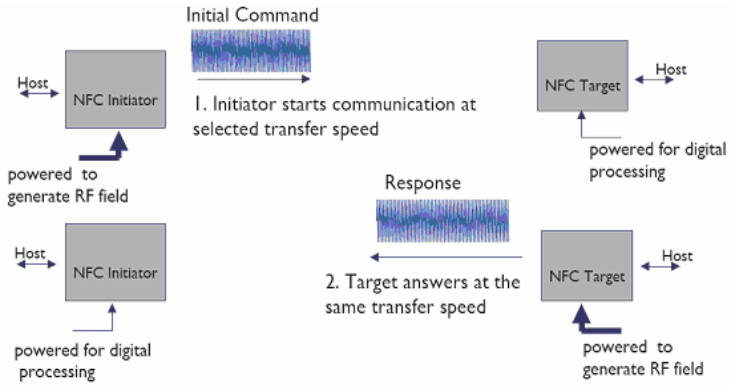

Ilustración 2. Modo activo de funcionamiento.

2.1.5 Usos y aplicaciones

La tecnología NFC está principalmente destinada a ser utilizado con teléfonos móviles. Hay tres principales casos de uso para NFC:

- tarjeta de emulación: el dispositivo NFC se comporta como una tarjeta de contacto existente
- modo lector: el dispositivo NFC está activo y permite leer una etiqueta RFID, por ejemplo para la publicidad interactiva.
- modo P2P: dos NFC dispositivos se comunican e intercambiar información.

Multitud de aplicaciones son posibles, tales como:

- Billetes de transporte en el móvil - una extensión de la infraestructura existente sin contacto y sin necesidad de poseer un billete físico propiamente dicho.

Ilustración 3. Pago en el transporte público, (1) en el autobús y en el metro (2).

- Pago a través del móvil - el dispositivo actúa como un débito / crédito de tarjetas de pago. Los usuarios podrán realizar transacciones económicas tan sólo acercando el dispositivo al terminal de cobro, sin tener que teclear nada en el móvil. Además, los teléfonos incorporarán una serie de características de seguridad para proteger los datos financieros y garantizar la seguridad de las comunicaciones.

Ilustración 4. Pago electrónico en comercios.

- Carteles inteligentes - el teléfono móvil se utiliza para leer las etiquetas RFID en carteles y pósters con el fin de obtener información sobre éste.

Ilustración 5. Lectura de información contenida en pósters, cuadros o carteles.

- Emparejamiento Bluetooth - en el futuro emparejamiento de los dispositivos Bluetooth con apoyo de NFC será tan fácil como que todos ellos esten juntos y acepten el proceso de emparejamiento. El proceso de activar Bluetooth en ambos lados, buscando, esperando, emparejamiento y la autorización será sustituido por un simple "contacto" de los teléfonos móviles.

Otras aplicaciones podrían incluir:

- Control de acceso físico a lugares o recintos.
- Control de acceso a una red informática.
- Pago en gasolineras.

- Inclusión de información médica para uso en emergencias.
- Apertura de vehículos.
- Documentos de identidad.
- Mobile Commerce.

- Domótica.
- Llaves NFC – para acceder a casa o a la oficina, llaves de habitaciones de un hotel, etc.

.

2.1.6 Comparación con otras tecnologías.

	NFC	RFID	IrDa	Bluetooth
Set -up time	<0.1ms	<0.1ms	~0.5s	~6 sec
Range	Up to 10cm	Up to 3m	Up to 5m	Up to 30m
Usability	Human centric Easy, intuitive, fast	Item centric Easy	Data centric Easy	Data centric Medium
Selectivity	High, given, security	Partly given	Line of sight	Who are you?
Use cases	Pay, get access, share, initiate service, easy set up	Item tracking	Control & exchange data	Network for data exchange, headset
Consumer experience	Touch, wave, simply connect	Get information	Easy	Configuration needed

2.1.7 Seguridad en NFC.

2.1.7.1 Escucha.

Como NFC es una interfaz de comunicación inalámbrica es evidente que la escucha es una cuestión importante. Cuando dos dispositivos se comunican a través de NFC que utilizan ondas de radiofrecuencia para hablar el uno al otro un atacante puede utilizar curso de una antena para recibir también la transmisión señales. Ya sea por la experimentación o investigación de la literatura el atacante puede tener el suficiente conocimiento sobre cómo extraer los datos transmitidos por la señal de RF recibida.

Además tanto el equipo receptor, así como al equipo decodificador se debe suponer que estará disponible para un atacante, ya que no hay equipos especiales necesaria.

La comunicación NFC normalmente se hace entre dos dispositivos próximos. Esto significa que no suele haber mas de 10 cm. (normalmente menos) de distancia entre uno y otro. La principal cuestión es cómo de próximo debe de situarse un atacante para ser capaz de recuperar una señal de RF. Lamentablemente, no existe una respuesta correcta a esta pregunta. La razón está en el gran número de parámetros que determinan la respuesta. Por ejemplo, la distancia depende de los siguientes parámetros, y hay muchos más.

- La característica de RF del dispositivo emisor (es decir, geometría de la antena, escudo del equipo, el PCB, el medio ambiente).
- Características de la antena del atacante (es decir, geometría de la antena, posibilidad de cambiar la posición en las 3 dimensiones).
- Calidad del receptor del atacante.
- Calidad del decodificador del atacante.
- Configuración de la ubicación donde el ataque se lleva a cabo (por ejemplo, barreras como paredes o metales).
- Potencia de envío del emisor.

Por consiguiente, cualquier número exacto dado sólo sería válido para un determinado conjunto de los anteriores parámetros y no puede ser utilizada para obtener directrices generales de seguridad.

Además, es de gran importancia el modo en que opera el remitente de los datos. Es decir, que si el remitente está generando su propio campo de RF (modo activo) o si el remitente está usando el campo de RF generado por otro dispositivo (el modo pasivo). En ambos casos usan una forma diferente de transmitir los datos y es mucho más difícil de realizar una escucha en los dispositivos que envían datos en modo pasivo.

Como se ha señalado anteriormente, no se puede dar unos números exactos para determinar como se hacen las escuchas, pero quizás este dato de distancia puede resultar intuitivo. Cuando un equipo envía datos en modo activo, la escucha se puede hacer hasta un distancia de unos 10 metros, mientras que cuando el envío se hace en modo pasivo, esta distancia se reduce significativamente a alrededor de 1 metro.

2.1.7.2 Corrupción de datos.

En lugar de sólo escuchar un atacante puede también tratar de modificar los datos que se transmite a través de la interfaz NFC. En el caso más sencillo el atacante sólo quiere perturbar la comunicación de tal forma que el receptor no sea capaz de comprender los datos enviados por los otros dispositivos.

La corrupción de datos puede lograrse mediante la transmisión de frecuencias válidas de los datos en el espectro en un determinado instante. Ese instante puede calcularse si el atacante tiene un buen conocimiento del sistema de modulación utilizado y codificación. Este ataque no es demasiado complicado, pero no permite que el atacante manipular los datos reales. Es básicamente un Ataque de Denegación de Servicio.

2.1.7.3 Modificación de Datos.

En la modificación de datos el atacante quiere que el dispositivo receptor reciba datos válidos pero manipulados. Esto es muy diferente a la corrupción de datos.

La viabilidad de este ataque es altamente dependiente de la amplitud de modulación. Esto se debe a que la decodificación de la señal es diferente para el 100% y 10% de modulación. Básicamente el atacante debe enviar señales de RF de tal manera que éstas se superpongan con las originales que recibiría el receptor.

2.1.7.4 Inserción de Datos.

Este tipo de ataque implica que el atacante inserta en el intercambio de datos entre dos dispositivos. Pero esto sólo es posible en caso de que el dispositivo necesite un tiempo muy grande para responder. El atacante podrá de esta manera enviar sus datos antes de la recepción válida. La inserción de será exitosa sólo si los datos son enviados antes de que el otro dispositivo comience con la respuesta. Si los dos flujos de datos se superponen, los datos se corromperán.

2.1.7.5 Man-in-the-Middle.

En el clásico Man-in-the-Middle Attack, dos partes que quieren hablar una con la otra (Alice y Bob), son engañados en una conversación a tres partes por un atacante Eve. Esto puede verse en la figura siguiente:

Ilustración 6. Ataque tipo man-in-the-middle.

Alice y Bob no deben ser conscientes del hecho de que no están hablando el uno al otro, pero ambos están enviando y recibiendo datos de Eve. Esta configuración es la clásica amenaza de clave no autenticada en clave en protocolos como Diffie-Hellmann. Alice y Bob quieren llegar a un acuerdo sobre una clave secreta, que luego utilizaran en un canal seguro. Sin embargo, como Eve se encuentra en el medio, tiene la posibilidad de establecer una clave con Alice y otra con Bob. Cuando mas tarde Alice y Bob hagan uso de esa clave, Eve es capaz de escuchar la comunicación y axial manipular los datos transferidos.

¿Cómo podría eso funcionar cuando el canal entre Alice y Bob es NFC?

Suponiendo que Alice utiliza el modo activo y Bob el modo pasivo, tenemos la siguiente situación: Alice genera el campo de RF y envía los datos a Bob. En el caso de Eva está suficientemente cerca, puede escuchar los datos enviados por Alice. Además ella debe perturbar la transmisión de Alice para asegurarse de que Bob no recibe los datos. Esto es posible para Eve, pero puede ser detectado por Alice. En el caso de Alice detecte esta interrupción, puede detener el protocolo. Ahora supongamos que Alice no comprueba que haya interrupciones activas, por lo que el protocolo puede continuar. En el siguiente paso a Eve necesita enviar datos a Bob. Eso es ya un problema, porque el campo RF generado por Alice todavía existe, por lo que Eve tendrá que generar un segundo campo RF. Esto, sin embargo, provocara la existencia de dos campos RF activos al mismo tiempo. Es prácticamente imposible alinear perfectamente estos dos campos, por lo tanto, es prácticamente imposible para Bob entender los datos enviados por Eve. Debido a esto y por la posibilidad de Alice para detectar el ataque llegamos a la conclusión de que en esta configuración de ataque es prácticamente imposible.

La única otra configuración posible es que tanto Alice como Bob usen el modo activo. En este caso, Alice enviará datos a Bob. Eva podrá interrumpir de nuevo la transmisión de Alice para asegurarse de que Bob no recibe los datos. En este punto Alice podría detectar la alteración realizada por Eva y poner fin al protocolo. Una vez más, vamos a suponer que Alice no hace este control y el protocolo continúa. En el siguiente paso Eve tendría que enviar datos a Bob. A primera vista esto parece mejor ahora, debido a la comunicación activa por ambos lados. Ahora Eve genera el campo RF y puede enviar datos, pero el problema es que ahora Alice puede escuchar porque se encuentra a la espera de una respuesta de Bob. En lugar de ésta, recibirá los datos enviados por Eve y de nuevo podrá detectar un problema en el protocolo y pararlo. Es imposible en esta configuración que Eve pueda enviar datos, ya sea a Alice o Bob, y asegurarse de que los datos no se reciben por Bob o Alice, respectivamente.

Por tanto, debido a las razones dadas anteriormente es prácticamente imposible realizar un ataque tipo Man-in-the-Middle escenario real.

2.2 Bluetooth

2.2.1 Introducción.

Norma que define un estándar global de comunicación inalámbrica de corto alcance que posibilita la transmisión de voz y datos entre dispositivos móviles (como teléfonos y ordenadores portátiles) y dispositivos de escritorio (como los ordenadores de sobremesa), mediante un enlace de radiofrecuencia seguro y globalmente libre.

Bluetooth surge principalmente buscando tres objetivos:

- Facilitar las comunicaciones entre equipos móviles y fijos.
- Eliminar cables y conectores entre éstos.
- Ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre nuestros equipos personales.

Esta tecnología de comunicación comprende hardware, software y requerimientos de interoperatibilidad. Para el establecimiento de la norma se creó en 1998 un grupo de interés especial (Special Interest Group) formado por las empresas Ericsson, IBM, Intel, Nokia y Toshiba.

2.2.2 Origen e historia.

El nombre de Bluetooth procede del rey vikingo Harald II apodado Blatan (o Bluetooth, como era conocido por los ingleses), el cual, durante la segunda mitad del siglo X, reino en Noruega y Dinamarca, y que recibió tal apodo debido a una rara enfermedad que daba color azul a su dentadura.

Fue un de los mas poderosos reyes de Europa y se caracterizó ser un buen comunicador y por unificar las tribus danesas, noruegas y suecas. De esta misma manera la tecnología Bluetooth intenta unificar las comunicaciones entre diferentes dispositivos como ordenadores y móviles.

El logo de Bluetooth es la unión de las runas nórdicas que representan respectivamente las letras H (hagall) y B (berkanan):

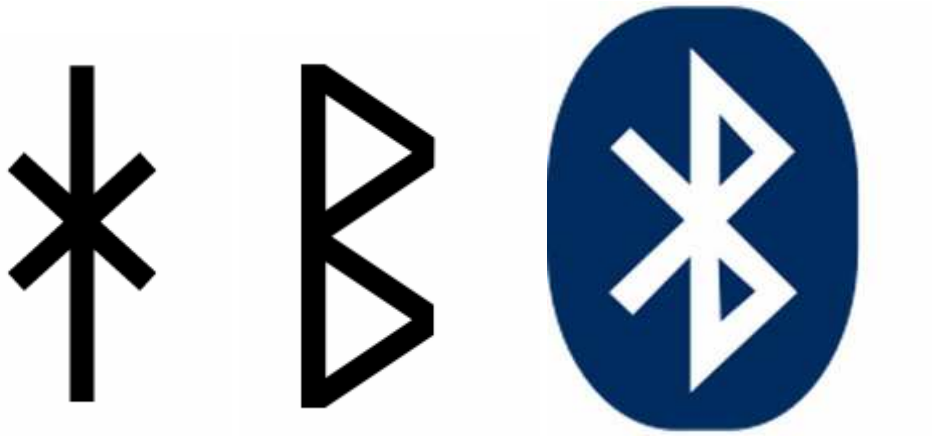

Ilustración 7. Runa Hagall, Berkanan y logo de Bluetooth

A pesar de que la primera versión o especificación de Bluetooth no fue publicada hasta 1999, ya desde 1994 Ericsson trabajaba en el estudio de un interfaz de radiofrecuencia, de bajo coste y bajo consumo, que permitiera la interconexión entre dispositivos móviles y otros accesorios sin la necesidad de cables. Mediante este estudio, Ericsson fue despertando el interés en otras empresas del sector de las Telecomunicaciones y se dieron cuenta que para que esta tecnología tuviera éxito era necesario que se asegurara la interoperabilidad entre dispositivos de diferentes fabricantes. Es así como surge en 1998 el Special Interest Group (SIG), formado por 5 promotores que fueron: Ericsson, Nokia, IBM, Toshiba e Intel. Más tarde se unirían a este grupo otras empresas como Microsoft, Compaq, Motorola, Dell, 3com, Axis Comunication, Xircom, Lucent.

Como muestra del gran éxito que fue alcanzando Bluetooth, el grupo SIG llego a estar formado por unas 1500 empresas adheridas en 2000 y siete años más tarde, en 2007, ya eran 9000 las empresas asociadas.

2.2.3 Principales características y especificaciones.

La tecnología Bluetooth opera en la banda de frecuencia 2,4GHz (mas en concreto entre 2,4 y 2,48Ghz), banda que comparte con otras tecnologías de radiofrecuencia (RF). Esta banda se caracteriza por no necesitar licencia para poder operar libremente en ella. Además, se utiliza la técnica de FHSS[1] (Frecuency Hoping Spread Spectrum) con un máximo de 1600 saltos/s. Los saltos de frecuencia se dan entre un total de 79 frecuencias con intervalos de 1Mhz; esto permite dar seguridad y robustez.

El rango máximo de transmisión usando BLUETOOTH es de unos 10 metros, pudiéndose ampliar hasta los 100 metros aumentando la potencia de transmisión o utilizando repetidores. Además, se alcanza una tasa de entre 720kbps y 1Mbps.

Clase	Potencia máxima permitida (mW)	Potencia máxima permitida (dBm)	Rango (aproximado)
Clase 1	100 mW	20 dBm	~100 metros
Clase 2	2.5 mW	4 dBm	~10 metros
Clase 3	1 mW	0 dBm	~1 metro

Ilustración 8. Tabla de rangos de transmision y potencias.

Con respecto a la seguridad, BLUETOOTH permite hace uso de técnicas de autentificación y encriptación (de 64 bits), para controlar la conexión y evitar que dispositivos puedan acceder a los datos o realizar su modificación. Así se definen diferentes modos de seguridad en el perfil de acceso: transmisión sin seguridad, con seguridad a nivel de servicio, o con seguridad a nivel de enlace (que resultaría la más potente). Más adelante se entra más en detalle en este aspecto, en concreto en el apartado 2.2.5**.**

Bluetooth permite una fácil integración con redes TCP/IP.

[1] Técnica de modulación en espectro ensanchado en el que la señal se emite sobre una serie de radiofrecuencias aparentemente aleatorias, saltando de frecuencia en frecuencia sincrónicamente con el transmisor. Los receptores no autorizados escucharán una señal ininteligible.

2.2.4 Arquitectura y funcionamiento.

El núcleo del sistema *Bluetooth* consiste en un transmisor de radio, una banda base y una pila de protocolos. El sistema permite la conexión entre dispositivos y el intercambio de distintos tipos de datos entre ellos.

2.2.4.1 Radio.

Bandas de frecuencia y organización de canales

Como ya se ha visto, el sistema Bluetooth funciona en la banda ISM de 2,4 GHz. Esta banda de frecuencias abarca 2400 - 2483,5 MHz. Los 79 canales RF se organizan por números, de 0 a 78, con un espacio de 1 MHz entre ellos, empezando por 2402 GHz.

Alcance reglamentario	Canales RF
2,400-2,4835 GHz	f=+k MHz, k=0,…,78

Para satisfacer la reglamentación relativa a las transmisiones fuera de banda de cada país, se utiliza una banda de guarda en los extremos inferior y superior de la gama de frecuencias.

Banda de guarda inferior	Banda de guarda superior
2 MHz	3,5 MHz

Modos de modulación.

Se definen dos modos de modulación. Un modo obligatorio, llamado modo de transferencia básica, que usa una modulación de frecuencia binaria para reducir al mínimo la complejidad del transmisor/receptor. Y un modo opcional, llamado de transferencia de datos mejorada, que usa modulación PSK y cuenta con dos variantes: π/4-DQPSK y 8DPSK.

Para la transmisión bidireccional se emplea una técnica de dúplex por división de tiempo (TDD) en ambos modos. Esta especificación define los requisitos de una radio Bluetooth, tanto para el modo de transferencia básica como de transferencia mejorada de datos.

Características del transmisor

Los dispositivos de clase 1 disponen de control de potencia. El control de potencia se usa para limitar la potencia transmitida por encima de +4 dBm. Por debajo de +4 dBm, la capacidad para controlar la potencia es opcional, y puede emplearse para controlar el consumo energético y el nivel global de interferencias.

Los dispositivos con capacidad para controlar la potencia usan comandos LMP para optimizar la potencia de salida en un enlace físico (consultar Protocolo de gestión de enlace en el apartado 2.2.4.3).

Emisiones espurias

Las emisiones espurias dentro de la banda se medirán con un transmisor de salto de frecuencia que transmitirá en un canal de radio y recibirá en otro; esto implica que el sintetizador puede cambiar de canal de radio entre recepción y transmisión,

pero siempre vuelve al mismo canal de transmisión. Este documento no hace referencia a las emisiones espurias fuera de la banda ISM, ya que el cumplimiento de la normativa en el país de destino del dispositivo es responsabilidad del fabricante.

Tolerancia de la radiofrecuencia

La precisión de la frecuencia central debe ser de ±75 KHz con respecto a la frecuencia central.

Transferencia de datos mejorada (EDR)

Un aspecto clave del modo de transferencia de datos mejorada es que se cambia la secuencia de modulación dentro del paquete. El código de acceso y la cabecera del paquete se transmiten dentro de la secuencia de modulación GFSK de 1 Mbps de la velocidad base, en tanto que la secuencia de sincronización, la carga útil y la secuencia de cola se transmite usando la secuencia de modulación de transferencia de datos mejorada.

Características de modulación EDR

Durante la transmisión del código de acceso y la cabecera del paquete, se usa la secuencia de modulación GFSK de transferencia básica. Durante la transmisión de la secuencia de sincronización, la carga útil y la secuencia de cola se usa un tipo de modulación PSK con una velocidad de transmisión de 2 Mbps, u, opcionalmente, 3 Mbps.

Características del receptor

Nivel de sensibilidad

El nivel de sensibilidad real se define como el nivel de entrada para el cual se satisface un porcentaje de error de bit (BER) del 0,1%. Para cualquier transmisor *Bluetooth*, la sensibilidad del receptor será de –70 dBm o inferior.

Rendimiento con interferencias

La interferencia co-canal y las interferencias en los canales adyacentes en 1 y 2 MHz se miden con la señal útil 10dB sobre el nivel de sensibilidad de referencia En el resto de canales de radiofrecuencia, la señal útil debe estar 3 dB sobre el nivel de sensibilidad de referencia.

Bloqueo fuera de banda

La supresión (o rechazo) fuera de banda se medirá con la señal útil 3 dB por encima del nivel de sensibilidad de referencia. La señal de la interferencia será de onda continua. El BER será ≤ 0,1%.

Características de intermodulación

La sensibilidad de referencia, BER = 0,1%, se satisfará bajo las siguientes condiciones:

- La señal útil tendrá una frecuencia f0 con un nivel de potencia de 6 dB por encima del nivel de sensibilidad de referencia.
- Una señal de onda senoidal estática tendrá una frecuencia f1 con un nivel de potencia de –39 dBm.

- Una señal con modulación *Bluetooth* tendrá una frecuencia f2 con un nivel de potencia de -39 dBm.

Transferencia de datos mejorada (EDR)
Nivel de sensibilidad EDR real

El nivel de sensibilidad real se definirá como el nivel de entrada para el cual se satisface un porcentaje de error de bit (BER) del 0,01%. El nivel de sensibilidad real de un receptor de transferencia de datos mejorada *Bluetooth* π/4-DQPSK y 8DPSK será de –70 dBm o superior. El receptor alcanzará el nivel de sensibilidad de –70 dBm con cualquier transmisor *Bluetooth* que cumpla los requisitos de la especificación del transmisor de transferencia de datos mejorada.

Rendimiento con interferencias

La interferencia co-canal y las interferencias adyacentes en los canales de 1 y 2 MHz se miden con la señal útil 10dB sobre el nivel de sensibilidad de referencia En el resto de frecuencias, la señal útil debe estar 3 dB sobre el nivel de sensibilidad de referencia.

2.2.4.2 Banda Base.

Descripción general.

La banda base *Bluetooth* es la parte del sistema *Bluetooth* que especifica o introduce los procedimientos de acceso de medios y capa física entre dispositivos *Bluetooth*. Dos o más dispositivos que comparten el mismo canal físico forman una piconet. Un dispositivo *Bluetooth* actúa como maestro de la piconet, y los demás dispositivos actúan como esclavos. Una piconet puede constar de hasta siete dispositivos activos. Además de estos dispositivos activos, la piconet puede contar con muchos más esclavos en estado de espera.

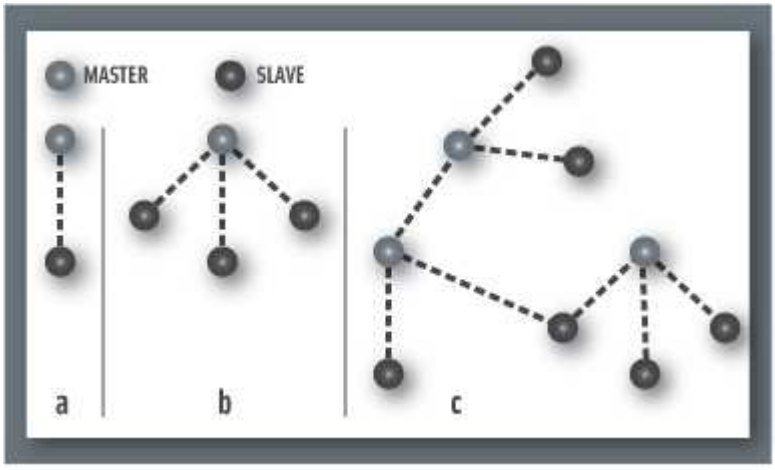

Ilustración 9. Piconets con un único esclavo (a), varios esclavos (b) y funcionamiento disperso (c).

Paquetes

Los datos se transmiten por radio en paquetes. El paquete general del modo de transferencia básica consta de tres entidades: el código de acceso, la cabecera, y la carga útil. La tasa de transferencia bruta es de 1 Mbps en el modo de transferencia básica.

Ilustración 10. Formato estándar de paquetes del modo de transferencia básica.

El paquete de transferencia de datos mejorada general consta de seis entidades: el código de acceso, la cabecera, el periodo de guarda, la secuencia de sincronización, la carga útil de la transferencia de datos mejorada, y la cola, o tráiler.

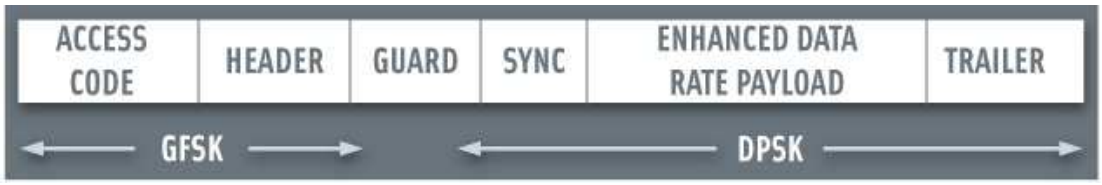

Ilustración 11. Formato estándar de paquetes para la transferencia de datos mejorada.

El código de acceso y el encabezamiento usan la misma secuencia de modulación que los paquetes del modo de transferencia básico, en tanto que la secuencia de sincronización, la carga útil de la transferencia de datos mejorada y la cola usan la secuencia de modulación de la transferencia de datos mejorada.

La tasa de transferencia de datos mejorada cuenta con una modalidad de modulación primaria que proporciona una velocidad de transmisión bruta de 2 Mbps, y una modalidad de modulación secundaria que proporciona una velocidad de transmisión bruta de 3 Mbps.

Reloj *Bluetooth*

Todos los dispositivos *Bluetooth* cuentan con un reloj nativo que debe derivarse de un reloj del sistema de libre funcionamiento. Para la sincronización con otros dispositivos se utilizan compensaciones (offsets) que, cuando se suman al reloj nativo, proporcionan relojes *Bluetooth* temporales sincronizados entre sí.

Dirección de dispositivos *Bluetooth*

A cada dispositivo *Bluetooth* se le asigna una dirección de dispositivo *Bluetooth* única de 48-bit (BD_ADDR) proporcionada por la autoridad reguladora de IEEE.

Códigos de acceso

En el sistema *Bluetooth*, todas las transmisiones que se realizan a través del canal físico se inician con un código de acceso. Se definen tres códigos diferentes:

- código de acceso del dispositivo (DAC)
- código de acceso del canal (CAC)
- código de acceso de consulta (IAC)

Canales físicos

Los canales físicos se definen mediante una secuencia de saltos pseudo aleatorios en los canales RF, la sincronización de los paquetes (ranuras) y un código de acceso. La secuencia de saltos se determina a partir de la dirección del dispositivo *Bluetooth* y de la secuencia de saltos seleccionada. La fase de la secuencia de saltos se determina mediante el reloj *Bluetooth*. Todos los canales físicos se subdividen en ranuras de tiempo cuya longitud depende del canal físico.

Canal físico básico de la piconet

La definición del canal físico básico de la piconet corresponde al maestro de la piconet. El maestro controla el tráfico del canal físico de la piconet mediante una secuencia de consultas.

Por definición, el dispositivo que inicia una conexión mediante una búsqueda de terminales hace las veces de maestro. Una vez establecida la piconet, es posible intercambiar las funciones de maestro y esclavo.

El canal físico básico de la piconet se divide en ranuras de tiempo de 625 µs de duración cada una.

Canal físico adaptado de la piconet

Los canales físicos adaptados de la piconet pueden usarse para dispositivos conectados con la función de salto adaptable de frecuencia (AFH) activada. Hay dos diferencias entre los canales físicos básicos y adaptados de la piconet. La primera es el mismo mecanismo del canal que hace que la frecuencia esclava sea la misma que la de la transmisión maestra precedente. La segunda diferencia es que el canal físico adaptado de la piconet puede no basarse en la totalidad de las 79 frecuencias del canal físico básico de la piconet.

Canal físico de detección de paginación

Si bien las funciones de maestro y esclavo no se definen antes de la conexión, se denomina "maestro" al dispositivo de paginación (que se convierte en maestro en el estado de CONEXIÓN) y esclavo al dispositivo que realiza la detección de la paginación (que actúa como esclavo en el estado CONEXIÓN).

El canal físico de detección de paginación sigue una secuencia de saltos más lenta que el canal físico básico de la piconet, y adopta una secuencia corta de saltos pseudo aleatorios entre los canales RF.

Canal físico de detección de búsqueda

Si bien las funciones de maestro y esclavo no se definen antes de la conexión, se denomina "maestro" al dispositivo de búsqueda y esclavo al dispositivo que realiza la detección de la búsqueda.

El canal físico de detección de búsqueda usa una secuencia de saltos más lenta que el canal físico de la piconet, y adopta una secuencia corta de saltos pseudo aleatorios entre los canales RF.

Enlaces físicos.

Un enlace físico representa una conexión de banda base entre dispositivos. Este tipo de enlace se asocia a tan sólo un canal físico. Los enlaces físicos tienen propiedades comunes que se aplican a todas las comunicaciones lógicas del enlace físico. Las propiedades comunes de los enlaces físicos son:

- Control de energía
- Supervisión de enlaces
- Cifrado
- Cambio de velocidad de transmisión dictado por la calidad del canal
- Control de paquetes en múltiples ranuras
- Comunicaciones lógicas

Pueden establecerse diferentes tipos de comunicaciones lógicas entre el dispositivo maestro y los dispositivos esclavos. Se han definido cinco comunicaciones lógicas:

- Comunicación lógica por conexión síncrona (SCO)
- Comunicación lógica por conexión síncrona ampliada (eSCO)
- Comunicación lógica por conexión asíncrona (ACL)
- Comunicación lógica por difusión del dispositivo esclavo activo (ASB)
- Comunicación lógica por difusión del dispositivo esclavo en espera (PSB)

Enlaces lógicos

Se definen cinco enlaces lógicos:

- Control del enlace (LC)
- Control ACL (ACL-C)
- Isócrono o asíncrono del usuario (ACL-U)
- Síncrono del usuario (SCO-S)
- Síncrono ampliado del usuario (eSCO-S)

Los enlaces lógicos de control LC y ACL-C se usan al nivel del control de enlaces y del gestor de enlaces, respectivamente. El enlace lógico ACL-U se utiliza para transmitir información síncrona o asíncrona del usuario. Los enlaces lógicos SCO-S y eSCO-S se usan para transmitir información síncrona del usuario. El enlace lógico LC se transmite en la cabecera del paquete, todos los demás enlaces lógicos se transmiten en la carga útil del paquete. Los enlaces lógicos ACL-C y ACL-U están indicados en el campo de la ID del enlace lógico (LLID) en la cabecera de la carga útil.

Los enlaces lógicos SCO-S y eSCO-S son transmitidos exclusivamente por las comunicaciones lógicas síncronas; el enlace ACL-U se transmite normalmente por la comunicación lógica ACL; sin embargo, también puede ser transmitido por los datos del paquete DV de la comunicación lógica SCO. El enlace ACL-C puede transmitirse por cualquiera de las dos comunicaciones lógicas SCO o ACL.

2.2.4.3 Pila de protocolos.

La especificación de Bluetooth establece que haya una unificación entre dispositivos de diferentes fabricantes y que así las diferentes aplicaciones puedan operar correctamente. Para ello es necesario que todos los dispositivos que vayan a participar en las comunicaciones ejecuten la misma pila de protocolos.

La pila de protocolos es la que se muestra en la siguiente figura, la cual podemos ver que esta formada por protocolos específicos de Bluetooth, como por ejemplo son Link Manager (LM) y Logical Link Control Adaption Protocol (L2CAP), así como por otros no específicos como pueden ser TCP, UDP, IP, OBEX, etc.

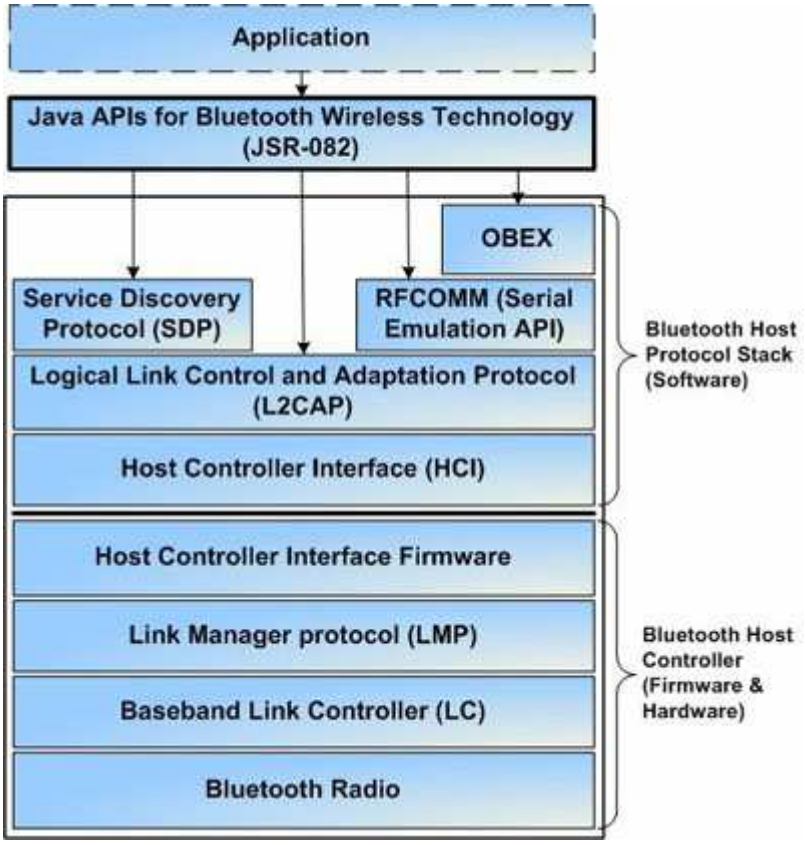

Ilustración 12. Pila de protocolos de Bluetooth.

Además de estos protocolos, se define el HCI (Host Controller Interface), encargado de proporcionar al controlador de banda base y al Link Manager de una interfaz de comandos que nos permite acceder a los registros de control y al estado del hardware.

Descripción de los protocolos.

Link Manager (LM).

Es el protocolo encargado de gestionar el establecimiento de la conexión entre los dispositivos, la autentificación y la configuración del enlace. El LM se encarga de localizar y comunicarse con otros gestores por medio del protocolo LMP (Link Manager Protocol), que en resumidas cuentas consiste en el envío de PDUs (Protocol Data Units) entre los dispositivos.

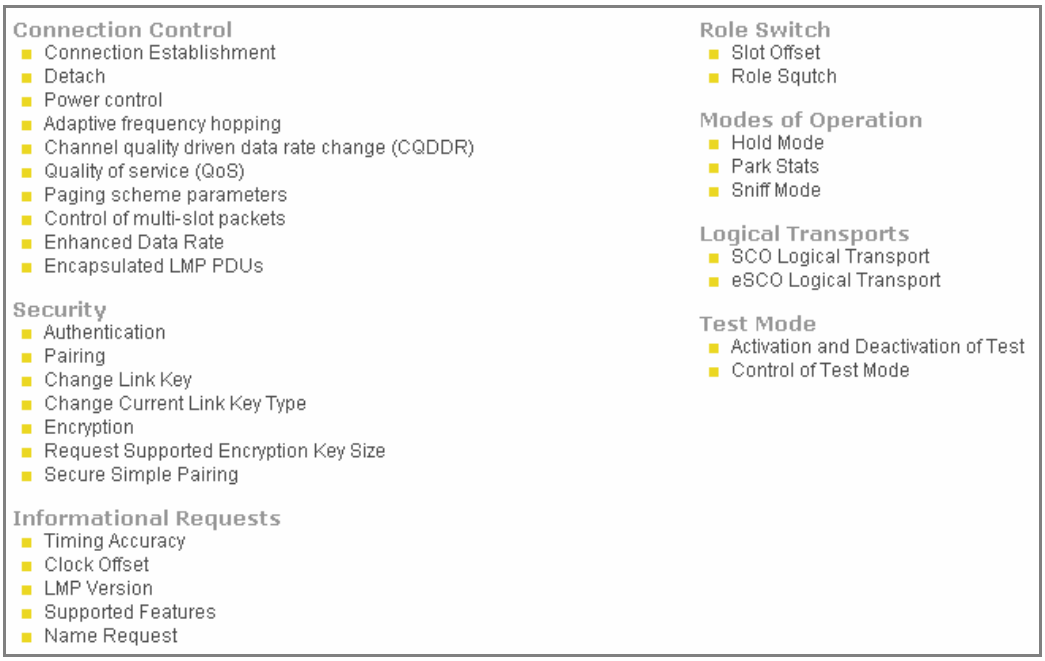

Ilustración 13. PDUs definidos en el protocolo LMP.

Una vez establecida la conexión, se procede a anunciar los servicios soportados:

- Transmisión y recepción de datos.
- Petición de nombre o ID (longitud máxima 16 caracteres)
- Petición de las direcciones de enlace.
- Establecimiento de la conexión.
- Autentificación.
- Negociación del modo de enlace y establecimiento, por ejemplo, modo datos o modo voz/datos. Esto puede cambiarse durante la conexión.

Host Controller Interface (HCI).

HCI proporciona una interfaz de comandos entre el controlador de la banda base y el gestor de enlaces, y acceso a los parámetros de configuración. Esta interfaz proporciona un método uniforme de acceso a todas las funciones de la banda base Bluetooth.

La capa HCI de la máquina intercambia comandos y datos con el firmware del HCI presente en el dispositivo Bluetooth. El driver de la capa de transporte de la controladora de la máquina (es decir, el driver del bus físico) proporciona ambas capas de HCI la posibilidad de intercambiar información entre ellas.

Una de las tareas más importantes de HCI que se deben realizar es el descubrimiento automático de otros dispositivos Bluetooth que se encuentren dentro del radio de cobertura. Esta operación se denomina en inglés inquiry (consulta). Es importante tener en cuenta que un dispositivo remoto sólo contesta a la consulta si se encuentra configurado en modo visible (discoverable mode).

El sistema Bluetooth proporciona una conexión punto a punto (con sólo dos unidades Bluetooth involucradas) o también una conexión punto multipunto. En el último caso, la conexión se comparte entre varios dispositivos Bluetooth.

Logical Link Control and Adaptation Protocol (L2CAP).

El protocolo L2CAP proporciona servicios de datos tanto orientados a conexión como no orientados a conexión a los protocolos de las capas superiores, junto con facilidades de multiplexación, segmentación y reensamblaje. L2CAP permite que los protocolos de capas superiores puedan transmitir y recibir paquetes de datos L2CAP de hasta 64 kilobytes de longitud.

L2CAP se basa en el concepto de canales. Un canal es una conexión lógica que se sitúa sobre la conexión de banda base. Cada canal se asocia a un único protocolo. Cada paquete L2CAP que se recibe a un canal se redirige al protocolo superior correspondiente. Varios canales pueden operar sobre la misma conexión de banda base, pero un canal no puede tener asociados más de un protocolo de alto nivel.

RFCOMM.

El protocolo RFCOMM proporciona emulación de puertos serie a través del protocolo L2CAP. Es un protocolo de transporte sencillo, con soporte para hasta 9 puertos serie RS-232 y permite hasta 60 conexiones simultáneas (canales RFCOMM) entre dos dispositivos.

Para los propósitos de RFCOMM, un camino de comunicación involucra siempre a dos aplicaciones que se ejecutan en dos dispositivos distintos (los extremos de la comunicación). Entre ellos existe un segmento que los comunica. RFCOMM pretende cubrir aquellas aplicaciones que utilizan los puertos serie de las máquinas donde se ejecutan. El segmento de comunicación es un enlace Bluetooth desde un dispositivo al otro (conexión directa).

RFCOMM trata únicamente con la conexión de dispositivos directamente, y también con conexiones entre el dispositivo y el modem para realizar conexiones de red. RFCOMM puede soportar otras configuraciones, tales como módulos que se comunican vía Bluetooth por un lado y que proporcionan una interfaz de red cableada por el otro.

OBEX (Object Exchange).

Es un protocolo de comunicaciones que facilita el intercambio de objetos binarios entre dispositivos. Es mantenido por la Infrared Data Association[2] (IrDA) pero ha sido adoptada también por el SIG de Bluetooh y por la Open Mobile Alliance[3] (OMA). Una de las primeras aplicaciones populares de OBEX tuvo lugar en la pda Palm III. Esta PDA y sus múltiples sucesoras utilizaron OBEX para intercambiar tarjetas de negocio, datos e incluso aplicaciones.

OBEX es similar en diseño y funcionalidad a HTTP, protocolo en el que el cliente utiliza un transporte fiable para conectarse a un servidor y así recibir o proporcionar objetos. No obstante, OBEX difiere en algunos puntos importantes:

- **Transporte:** HTTP funciona normalmente sobre un puerto TCP/IP. OBEX, en cambio, es comúnmente implementado sobre una pila IrLAP/IrLMP/Tiny TP de

[2] IrDA define un estándar físico en la forma de comunicación (transmisión y recepción) de datos por rayos infrarrojos. IrDA se crea en 1993 entre HP, IBM, Sharp y otras empresas.

[3] **OMA**) es una organización que desarrolla estándares abiertos para la industria de la telefonía móvil.

un dispositivo de infrarrojos; mientras que funcionando con Bluetooth, se suele implementar sobre una pila en Banda Base/Link Manager/L2CAP/RFCOMM. En cualquier caso, ofrece otras posibilidades.

- **Transmisiones binarias:** P ara intercambiar información sobre una petición o un objeto, HTTP utiliza texto legible por el ser humano, mientras que OBEX utiliza tripletes binarios llamadas cabeceras. Éstos, resultan más simples de elaborar para dispositivos con características limitadas.

- **Soporte para realizar sesiones:** Las transacciones HTTP carecen inherentemente de estado. Generalmente, un cliente HTTP establece una conexión, efectúa una sola petición, recibe respuesta y cierra la conexión. En OBEX, una sola conexión de transporte podría utilizarse para efectuar varias operaciones relacionadas entre sí. De hecho, las últimas novedades de la especificación OBEX permiten almacenar la información del estado de una conexión intacta incluso si la conexión finalizó inesperadamente.

Protocolo de Descubrimiento de Servicios (SDP).

Este protocolo permite a las aplicaciones cliente descubrir la existencia de diversos servicios proporcionados los servidores de aplicaciones, así como el conjunto de los atributos y propiedades de éstos. Estos atributos de servicio incluyen el tipo o clase de servicio ofrecido y el mecanismo o la información necesaria para utilizar dichos servicios.

El servidor mantiene una lista de registros de servicios, los cuales describen las características de los servicios ofrecidos. Cada registro contiene información sobre un determinado servicio. Un cliente puede recuperar la información de un registro de servicio almacenado en un servidor SDP lanzando una petición SDP. Si el cliente o la aplicación asociada con el cliente deciden utilizar un determinado servicio, debe establecer una conexión independiente con el servicio en cuestión. SDP proporciona un mecanismo para el descubrimiento de servicios y sus atributos asociados, pero no proporciona ningún mecanismo ni protocolo para utilizar dichos servicios.

Normalmente, un cliente SDP realiza una búsqueda de servicios acotada por determinadas características. No obstante hay momentos en los que resulta deseable descubrir todos los servicios ofrecidos por un servidor SDP sin que pueda existir ningún conocimiento previo sobre los registros que pueda contener. Este proceso de búsqueda de cualquier servicio ofrecido se denomina navegación o browsing.

2.2.5 Seguridad en Bluetooth.

En el mundo inalámbrico en que vivimos se transmiten datos continuamente entre diferentes dispositivos sin que nos percatemos de ello. Es muy importante recalcar que estas transmisiones deben realizarse de forma segura y es por eso que desde el principio de la especificación de Bluetooth se ha hecho hincapié en la seguridad de las conexiones entre dispositivos.

2.2.5.1 Modos de seguridad.

Los desarrolladores de productos equipados con tecnología inalámbrica Bluetooth tienen diversas opciones a la hora de poner en práctica mecanismos de protección. Existen tres modos de seguridad para las conexiones entre dos dispositivos:

- Modo de seguridad 1: no seguro. Todos los mecanismos de seguridad (autenticación y cifrado) están deshabilitados. Además el dispositivo se configura en modo promiscuo, permitiendo que todos los dispositivos Bluetooth se conecten a él.
- Modo de seguridad 2: seguridad impuesta a nivel del servicio (L2CAP). Los procedimientos de seguridad son inicializados después de establecerse un canal entre el nivel LM y el de L2CAP. Un gestor de seguridad controla el acceso a servicios y dispositivos. Variando las políticas de seguridad y los niveles de confianza se pueden gestionar los accesos de aplicaciones con diferentes requerimientos de seguridad que operen en paralelo. Su interface es muy simple y no hay ninguna codificación adicional de PIN o claves.
- Modo de seguridad 3: seguridad impuesta a nivel del enlace. Los procedimientos de seguridad son iniciados antes de establecer algún canal. Aparte del cifrado tiene autenticación PIN y seguridad MAC. Su metodología consiste en compartir una clave de enlace secreta entre un par de dispositivos. Para generar esta clave, se usa un procedimiento de “pairing” cuando los dos dispositivos se comunican por primera vez.

Es el propio fabricante de cada producto quien determina el modo de seguridad del mismo. Tanto los dispositivos como los servicios que ofrece, cuentan con distintos niveles de seguridad.

Los niveles para los dispositivos son dos: "dispositivo de confianza" (dispositivo que ya ha sido emparejado con uno de sus dispositivos, y que tiene acceso sin restricción a todos los servicios) y "dispositivo poco fiable".

Los servicios cuentan con tres niveles de seguridad:
- servicios que precisan autorización y autenticación.
- servicios que solo precisan autenticación.
- servicios abiertos a todos los dispositivos.

2.2.5.2 Emparejamiento de dispositivos.

Por defecto, la comunicación Bluetooth no se valida, lo que implica que cualquier dispositivo puede en principio establecer una conexión con cualquier otro. Aun así un dispositivo Bluetooth (por ejemplo un teléfono móvil) puede solicitar autenticación para realizar un determinado servicio.

La autenticación de Bluetooth normalmente se realiza utilizando códigos PIN. Un código PIN no es mas que una cadena ASCII de hasta 16 caracteres de longitud, de tal manera que los usuarios deben introducir el mismo código PIN en ambos dispositivos para poder emparejarlos (pairing). Una vez que el usuario ha introducido el PIN adecuado ambos dispositivos generan una clave de enlace. Una vez generada, la clave se puede almacenar en el propio dispositivo o en un dispositivo de almacenamiento externo. La siguiente vez que se comuniquen ambos dispositivos se reutilizará la misma clave.

Es importante recordar que si la clave de enlace se pierde en alguno de los dispositivos involucrados se debe volver a ejecutar el procedimiento de emparejamiento.

No existe ninguna limitación en los códigos PIN a excepción de su longitud. Algunos dispositivos pueden obligar a escribir un número predeterminado de caracteres para el código PIN.

2.2.5.3 Inicialización y generación de la claves.

La clave de enlace es generada durante la fase de inicialización, cuando dos dispositivos empiezan a comunicarse, es decir, cuando los usuarios introducen un PIN idéntico en ambos dispositivos. Después de completarse la inicialización, los dispositivos se autentican de manera automática y transparente y se lleva a cabo el cifrado de la conexión. A partir del código PIN, se obtiene la clave de enlace común a través del siguiente algoritmo:

1) Se genera una clave de inicialización común Kinit de 128 bits usando el algoritmo E22 a partir del código de seguridad Bluetooth (PIN), la longitud del mismo, la dirección BD_ADDR de 48 bits y un número aleatorio IN_RAND de 128 bits.

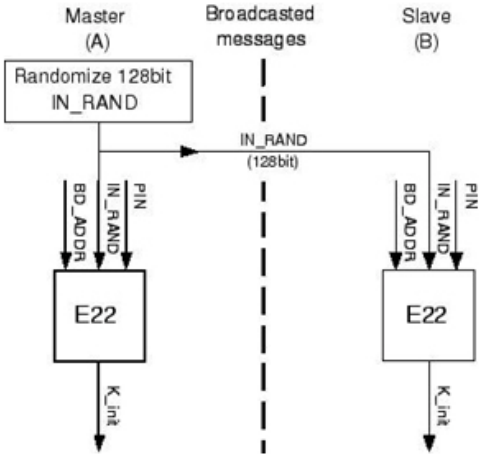

Ilustración 14. Generación de la clave de enlace.

2) Se genera la clave de enlace Kab usando el algoritmo E21. Los dispositivos usan la clave de inicialización Kinit para intercambiar dos nuevos números aleatorios, conocidos como LK_RAND A y LK_RAND B. Cada dispositivo genera un número aleatorio y se lo envía al otro dispositivo previamente XOReado bit a bit con Kinit. Dado que ambos dispositivos conocen Kinit, cada dispositivo conoce ambas LK_RAND. A partir de la dirección BD_ADDR y LK_RAND, se genera la clave de enlace Kab.

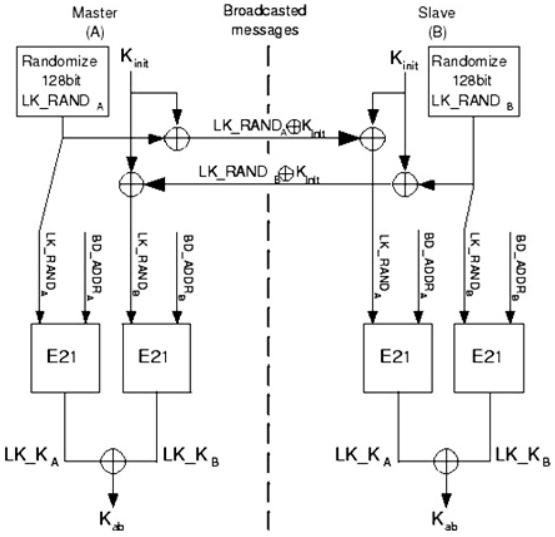

Ilustración 15. Generación de la clave Kab.

3) Una vez que los dispositivos emparejados disponen de la clave de enlace Kab, utilizan esta clave común para autenticarse automáticamente en las sucesivas conexiones.

Una vez que los dispositivos emparejados disponen de la clave de enlace, utilizan esta clave común para autentificarse automáticamente en las sucesivas conexiones. El proceso de autenticación está basado en el esquema desafío/respuesta y transcurre de la siguiente forma:

1) El dispositivo demandante envía su dirección BD_ADDR al dispositivo verificador.

2) El verificador devuelve un desafío aleatorio de 128 bits al demandante.

3) El demandante usa el algoritmo E1 para generar la respuesta de autenticación (SRES) de 32 bits, usando como parámetros de entrada la dirección BD_ADDR del demandante, la clave de enlace Kab almacenada y el desafío. El verificador realiza la misma operación en paralelo.

4) El demandante devuelve la respuesta SRES al verificador.

5) El verificador comprueba la respuesta SRES recibida con la respuesta SRES calculada por él.

6) Si los valores de SRES coinciden, el verificador y el demandante intercambian los papeles y repiten el proceso entero.

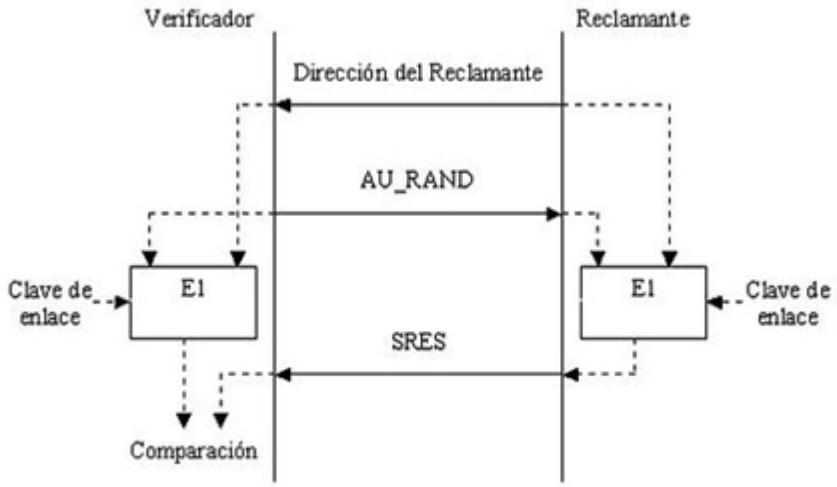

Ilustración 16. Proceso de autenticación challenge-response.

La especificación de Bluetooth establece que si se produce un fallo durante el proceso de autenticación, y para prevenir que un atacante pruebe claves de enlace aleatorias en un ataque de fuerza bruta, debe transcurrir cierto período de espera antes de que se pueda llevar a cabo un nuevo intento de autenticación. Para cada sucesivo intento fallido, el tiempo de espera aumenta exponencialmente.

En este proceso, se crea además un número de 96 bits llamado ACO (Authenticated Ciphering Offset) en ambos dispositivos, que será usada durante para la creación de la clave de cifrado.

2.2.5.4 Cifrado de datos.

El cifrado de datos protege la información que se transmite en un enlace entre dispositivos Bluetooth. Garantiza la confidencialidad del mensaje transmitido, de forma que si un paquete es capturado por un usuario que no posea la clave de descifrado, el mensaje le resultará ininteligible.

Su implementación es opcional, pero necesita que se haya producido anteriormente una autenticación. El maestro y el esclavo deben ponerse de acuerdo en utilizar cifrado o no. En caso afirmativo, deben determinar el tamaño de la clave de cifrado, para lo cual, maestro y esclavo intercambian mensajes hasta alcanzar un acuerdo.

No siempre es posible llegar a un acuerdo sobre el tamaño de la clave, en este caso se indica a las unidades Bluetooth que no se les permite comunicarse utilizando cifrado en el enlace.

Tras esta negociación comienza el proceso de cifrado. El maestro genera una clave de cifrado KC de 128 bits usando el algoritmo E3, el cual requiere como parámetros de entrada un número aleatorio de 128 bits, la clave de enlace Kab generada durante el procedimiento de emparejamiento y número COF (Ciphering Offset) de 96 bits basado en el valor temporal ACO calculado durante el procedimiento de autenticación.

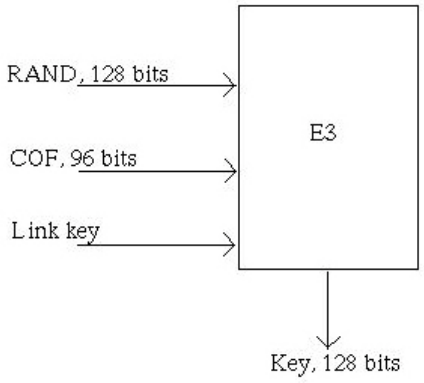

Ilustración 17. Generación de la clave de cifrado.

Una vez que la clave de cifrado se ha generado con éxito, el maestro se encuentra en condiciones de transmitir datos cifrados, para lo cual debe detener temporalmente el tráfico de datos de los niveles superiores y así evitar la recepción de datos corruptos.

2.2.6 Principales usos y aplicaciones.

Hoy en día, la aparición de ha permiten cambiar radicalmente la forma en la que los usuarios interactúan con los teléfonos móviles y otros dispositivos, dando lugar así a un gran numero de aplicaciones y usos de esta tecnología.

Desde el punto de vista de los usuarios, Bluetooth supone una tecnología que permite una comunicación fácil, instantánea, rápida, en cualquier lugar y que además su coste es bajo; sin olvidar su impacto en la forma de realizar los procesos, al sustituir los medios convencionales y posibilitar nuevos negocios y aplicaciones.

La aplicación de esta tecnología se puede percibir desde la implementación de una red inalámbrica hasta la posibilidad de transferir una fotografía de una cámara a un móvil para enviarla por correo electrónico o transferirla a la impresora para imprimirla en ausencia de cables. Desde el punto de vista profesional, la aplicación más práctica, es la posibilidad de montar una red inalámbrica en salas o entornos que ofrezcan dificultades para montar una LAN convencional. Para ello se utiliza un punto de acceso y cada puesto lleva instalado una PC Card con esta tecnología. A continuación se detallan algunas de las aplicaciones más importantes de Bluetooth:

- Transferencia de archivos: El servicio consiste en la transferencia de archivos .doc, .xls, .ppt, .wav, y .jpg, carpetas y directorios de un dispositivo a otro. Además ofrece la posibilidad de ver el contenido de carpetas de dispositivos remotos.

- Escritorio Inalámbrico: Bluetooth nos ofrece la posibilidad de eliminar los cables que utilizamos en nuestros equipos: desde un teclado inalámbrico, pasando por el ratón, incluso un disco duro portátil que se comunique mediante esta tecnología.

- Conexión a Internet: Esta aplicación permite conexión inalámbrica, un usuario tiene acceso a Internet mediante un teléfono móvil o mediante un módem inalámbrico, tal cual como si fuera una línea telefonía fija.

- Acceso Inalámbrico a LAN: En este servicio, múltiples equipos terminales de datos usan puntos de acceso LAN, llamados LAP (LAN access point), como una conexión inalámbrica a una red de área local LAN. Una vez conectados, funcionan como si estuvieran conectados a una LAN vía red.

- Sincronización Automática: El servicio consiste en sincronizar automáticamente y de manera continua la información PIM (Personal Information Managament) en los distintos dispositivos Bluetooth; básicamente la información es la concerniente a calendario, lista de direcciones y teléfonos, mensajes y notas.

- Teléfono tres en uno: Un mismo teléfono se puede utilizar como fijo, si se encuentra dentro del radio de acción del punto de acceso instalado, como teléfono móvil si nos encontramos fuera de radio de acción del punto de acceso, y por último, como medio de acceso a nuestros contactos, teléfonos, correo electrónico, etc.

- Dispositivo Manos Libres Inalámbrico: El dispositivo manos libres puede conectarse de manera inalámbrica al teléfono móvil, al ordenador portátil u otro móvil, con el fin de actuar como un dispositivo remoto con entrada y salida de audio. Como se puede apreciar, las aplicaciones de Bluetooth son casi infinitas y permiten cambiar radicalmente la forma en la que los usuarios interactúan con los teléfonos móviles y otros dispositivos. Una de las primeras compañías en lanzar un producto Bluetooth ha sido Ericsson.

Alianzas como las de Nokia y Fuji permitirán a los propietarios de cámaras digitales hacer fotos para luego transmitirlas a través del móvil a la impresora situada en casa o al disco duro del ordenador. Mientras, compañías como Motorola y JVC desarrollan continuamente tecnologías aún más avanzadas que harán estos avances extensibles al vídeo o al DVD. La compañía Sony ha hecho posible la implantación de microchips Bluetooth en toda su gama de productos. En sólo un par de años

caminaremos escuchando música en un reproductor MP3 mientras descargamos nuevas canciones y actualizamos el repertorio musical a través del móvil. La utilidad de Bluetooth sólo está delimitada por la imaginación de los ingenieros y los usuarios.

Entre otras aplicaciones, Bluetooth permite conectar cámaras de vigilancia, servir con mandos a distancia, permite utilizar un teléfono celular como inalámbrico, para abrir puertas, conectar electrodomésticos, pasar ficheros MP3 del móvil al PC, y por supuesto, para conectar todo tipo de dispositivos a Internet, formando puntos de acceso. Encuentra aplicación en la industria de automoción, en medicina para monitorización de los enfermos sin necesidad de tener cables conectados a su cuerpo, automatización del hogar, lectura de contadores, asociado a un lector de código de barras.

En definitiva, Bluetooth se esta convirtiendo en una tecnología de uso cotidiano, y sus características le han permitido ser utilizado en numerosos campos de aplicación, y muchos mas que llegaran en un futuro.

2.3 J2ME.

2.3.1 Introducción.

En este apartado vamos a ver la edición de la plataforma Java que Sun Microsistems ha diseñado específicamente para dispositivos móviles y embebidos, Java 2 Micro Edition (J2ME). Aunque en esta introducción veamos una visión general de J2ME y su entorno, nos centraremos en el estudio de las herramientas y bibliotecas que ofrece J2ME para la programación de dispositivos específicos como teléfonos móviles y PDAs.

Java comenzó su andadura como lenguaje de programación a mediados de la década de los noventa del siglo pasado. Originalmente fue concebido como un lenguaje de programación que permitía escribir un programa una vez y poder ejecutarlo en multitud de ordenadores, con diferentes plataformas sin tener que compilarlo de nuevo. Esa es una gran ventaja y una característica muy deseable en el entorno de los pequeños dispositivos, por lo que se ha exportado esa filosofía a estos aparatos. Así, mediante J2ME se podrán escribir aplicaciones para una gran variedad de dispositivos diferentes.

Por supuesto, esta nueva edición de Java no es la misma que se utiliza para desarrollar aplicaciones distribuidas en Internet, por ejemplo, sino que es una versión reducida que se adapta claramente a las características físicas de los pequeños dispositivos.

La primera versión salió allá por 1999, la cual sólo contenía una única máquina virtual y un único API, hecho que puso de manifiesto la insuficiencia de esta solución para la gran variedad de dispositivos diferentes. De esta forma, en el año 2000, nació la primera versión de una configuración, es decir, el Connected Limited Device Configuration (CLDC 1.0). Una configuración ofrece el API básico para programar dispositivos, aunque no aporta todas las clases necesarias para desarrollar una aplicación completa. Por tanto, la primera configuración no tenía las herramientas necesarias para permitir a los desarrolladores escribir programas para dispositivos. En julio de 2000 nació la primera implementación de un perfil, concretamente el llamado Mobile Information Device Profile (MIDP), aunque no estaba destinado a PDAs sino a teléfonos móviles y a paginadores.

Mas tarde, saldría la versión 2.0 de MIDP haciendo que J2ME se consolidase más aun dentro de la comunidad de desarrolladores de dispositivos móviles, y expandiéndose a una gran velocidad hasta nuestros días.

2.3.2 Diferencias con J2EE/J2SE.

Algunas diferencias que ofrece J2ME con respecto a J2EE y J2SE, directamente derivadas de las condiciones en las que se va a hacer uso de esta edición, son las siguientes:

- Tipos de datos: J2ME no incluye los tipos float y double, ya que la mayoría de los dispositivos CLDC no tienen unidad de coma flotante debido fundamentalmente a que es una operación muy costosa.

- Preverificación: La verificación del código en J2ME se hace fuera del dispositivo, con objeto de reducir la carga de la máquina.

- Inclusión de los ficheros "descriptor" y "manifiesto" al empaquetar ficheros J2ME, conteniendo información sobre las aplicaciones que incluyen.

- Nueva biblioteca gráfica adaptada a los dispositivos con memorias de poco tamaño y pantallas también pequeñas.

- No existe un método main como entrada para la ejecución de la función. Éste se sustituye por el método "start app".

- La recolección de basura se hace de manera manual y no automática como en el J2EE. Esta decisión se toma así para reducir la utilización de la memoria.

2.3.3 Arquitectura.

Como ya se ha visto en la introducción, dos son los pilares básicos en J2ME, la configuración y el perfil.

2.3.3.1 Configuraciones.

J2ME presenta dos configuraciones: CLDC y CDC. La primera se dedica a dispositivos con estrictas limitaciones de memoria, capacidad de cálculo, consumo y conectividad de red. Por otro lado, CDC se encarga de dispositivos con más potencia. Parte de CLDC es un subconjunto de CDC, por lo que la portabilidad de aplicaciones se puede conseguir cuando nos movemos de un entorno más restringido a otro más rico. De la misma manera, y siguiendo en el hilo de la portabilidad, una aplicación en J2ME podrá ejecutarse en J2SE normalmente, salvo que se utilicen las bibliotecas específicas de J2ME.

CLDC

Veamos más detalladamente algunas características de CLDC, ya que es la configuración que se ha usado en este proyecto. Comencemos por las propiedades mínimas requeridas a un dispositivo para poder desarrollar con esta configuración:

- De 160 a 512 KB de memoria disponible para el entorno de Java.
- Un procesador de 16 o 32 bits.
- Consumo de energía bajo.
- Permiten algún tipo de conectividad a una red

Al tener como objetivo dispositivos con prestaciones reducidas, CLDC elimina una gran cantidad de características que sí aparecen en J2EE y J2SE, tanto en el propio lenguaje Java como en la máquina virtual, como por ejemplo:

- Interfaz nativo de Java (Java Native Interface -JINI) (Máquina virtual).
- Cargadores de clases definidas por el usuario (Máquina virtual).
- Grupos de hilos e hilos demonios (Máquina virtual).
- Finalización (lenguaje Java).
- Referencias débiles (Máquina virtual).
- Reflexión (Máquina virtual).
- Tipos de datos de punto flotante (lenguaje Java).
- Algunos aspectos de seguridad y APIs (Máquina virtual).
- Verificación de ficheros de clases (Máquina virtual).
- Posee algunas limitaciones en las gestiones de errores (lenguaje Java).

Pero ¿qué razones se consideraron para eliminar esas prestaciones? Por supuesto, una de ellas se basa en cuestiones de ahorro de memoria, ya que el tamaño general del API queda reducido. Aunque otras también han sido quitadas por cuestiones de aligerar el procesador, como es el caso de las operaciones en punto flotante, o la verificación de las clases. Concretamente, esta operación, que identifica y rechaza ficheros de clases inválidas, se realizaba en la máquina virtual en la edición J2SE, pero en ese caso aumenta su tamaño. Por esta razón, en J2ME la verificación se hace en dos partes: la primera, la preverificación, en un ordenador distinto; la segunda sí se lleva a cabo en el propio dispositivo, pero en este caso es mucho más simple y rápida. Más adelante estudiaremos de manera más detallada el proceso de verificación.

Con respecto a la seguridad, al no definir CLDC completamente el sistema de seguridad de Java deben eliminarse prestaciones que sí figuran en el J2SE y que harían muy vulnerables las aplicaciones. Por ejemplo, sin este modelo de seguridad, un cargador de clases definido por el usuario podría alterar la forma en que el camino de las clases fuera recorrido, pudiendo una aplicación sustituir trozos de las bibliotecas del núcleo de Java y ganar acceso al dispositivo de tal forma que pudiera dañarlo.

También se considera la simple conveniencia como criterio: algunas clases pueden ser desarrolladas por los programadores basadas en otras que sí se mantienen. Igual ocurre con algunos métodos de clases. Por otro lado, otras clases se eliminan por que no tiene razón de ser.

En cuanto a la máquina virtual de CLDC, KVM, requiere entre 40 y 80 Kbytes dependiendo de las opciones de compilación y el tipo de dispositivo para el que se compile. Esto implica que se podrán ejecutar aplicaciones con un total de 128 Kbytes. Aparte de esto, se necesitan otros 32 Kbytes para memoria dinámica de la aplicación a ejecutar. KVM está implementada en C y está diseñada para ser tan completa y rápida como sea posible. De hecho, puede ejecutarse de un 30 a un 80% de la velocidad de la JVM.

Volviendo a la verificación de clases, la máquina virtual de Java estándar efectúa un proceso en tiempo de ejecución que se denomina verificación de clases, el cual se lleva a cabo antes de cargar ninguna clase en memoria. El objetivo es asegurar la integridad de los ficheros donde se almacena una clase Java y que el código en ella no intente acceder a memoria fuera de su espacio de nombres, eliminando la posibilidad de que pueda sustituir alguno de los paquetes del núcleo de Java poniendo así en juego la seguridad del sistema. Esta etapa juega un papel muy importante en el modelo de seguridad de Java.

Para que nos hagamos una idea, J2SE verifica, entre otros, estos puntos:

- Inicialización de todas las variables locales antes de su uso.
- El constructor de un objeto debe ser llamado justo seguidamente de la creación del mismo, y antes de que se use.
- Cada constructor tiene que comenzar con una llamada al constructor de su superclase.
- Las variables locales y miembros estáticos deben contener referencias a objetos que sean asignables legalmente.

Si nos trasladamos a CLDC, este proceso será muy costoso en términos de uso de recursos, ya que requiere mucha memoria, procesador y espacio para código binario. Es por esto por lo que los diseñadores de KVM decidieron hacer la verificación de clases de manera diferente a como se hace con JVM. Así, antes de que

la clase se llegue a emplear en el dispositivo, ésta es modificada externamente por una utilidad "preverificadora". La idea es añadir al fichero clase generado por javac nuevo código que identifique la clase como válida (pasa a ser una clase verificada). Seguidamente, se transfiere al dispositivo y la KVM sólo tiene que comprobar si esta información está o no presente o contiene o no la información correcta. En cualquiera de los dos casos negativos, el proceso de carga se interrumpe y se lanza una excepción. Esta comprobación se puede hacer justo cuando se carga la clase o como parte del proceso de instalación de la aplicación. En cualquier caso es un proceso más rápido que la preverificación y requiere menos memoria.

2.3.3.2 Perfiles.

Los perfiles suministran un interfaz de usuario, métodos de entrada y mecanismos de persistencia, así como un entorno de desarrollo completo para un conjunto específico de dispositivos, soportando sus características concretas. Los perfiles son necesarios porque las configuraciones no presentan ninguna de estas prestaciones. Así, J2ME oferta al programador el concepto de perfil, el cual define una plataforma común para un grupo determinado de dispositivos, los cuales comparten características y funciones. Un dispositivo puede cubrir más de un perfil.

Los perfiles se asientan sobre una configuración dada y utilizan los servicios que ofrece, aunque es la parte de la arquitectura J2ME que más cerca se encuentra del aparato físico. Para el caso de CLDC, su único perfil es MIDP, el cual cubre el mercado de dispositivos móviles, que comparten características como memoria limitada, pantalla pequeña, conexión a algún tipo de red inalámbrica y mediante banda ancha limitada y alimentados normalmente por baterías.

MIDP - MIDlets

Las aplicaciones MIDP se denominan "MIDlets", las cuales pueden utilizar tanto las facilidades aportadas por MIDP como las APIs que MIDP hereda de CLDC, pero nunca acceden directamente al sistema operativo subyacente, por lo que no serían portables. Un MIDlet consiste en una clase Java, como mínimo, derivada de la clase abstracta MIDP, y que se ejecutan en un entorno de ejecución dentro de la máquina virtual, la cual provee un ciclo de vida bien definido controlado mediante métodos que cada MIDlet debe implementar. Un grupo de MIDlets que están relacionados se suelen agrupar en un MIDlet suite. Todos estos MIDlet se empaquetan, instalan, desinstalan y borran como una única entidad y comparten recursos tanto en tiempo de ejecución como estáticos.

Comenzando con los requerimientos de memoria, MIDP necesita 128 KB de RAM disponible para la implementación correspondiente. A esta cantidad debemos sumarle la que necesita CLDC y como mínimo 32 Kbytes para almacenar la pila de la aplicación, tamaño que obliga al programador a tener bastante cuidado a la hora de diseñar las aplicaciones. Además, los dispositivos MIDP cuentan con 8 KB como mínimo de memoria no volátil que se utiliza como almacenamiento persistente, que no se borra tras apagar el aparato (salvando el problema del cambio de batería).

Sobre las pantallas de los dispositivos, la especificación MIDP indica que ésta requiere 96 pixels de ancho por 54 de alto y que debe soportar al menos dos colores.

Con respecto a los tipos de entradas del dispositivo, el rango es muy amplio: desde los que tienen un teclado alfanumérico completo, hasta aquellos que permiten escribir en ciertas áreas de la pantalla, pasando por los teclados de los teléfonos móviles. La especificación mínima requiere un teclado que permita marcar los

números del 0 al 9, junto con el equivalente a las teclas del cursor y un botón de selección.

MIDP no asume que los dispositivos estén permanentemente conectados a una red, ni siquiera que soportan TCP/IP, pero que sí tienen algún tipo de acceso a una red. En este sentido, la especificación sí establece que soporte HTTP 1.1, bien mediante una pila de protocolos o una pasarela WAP.

También los sistemas operativos de los dispositivos tienen restricciones. Por ejemplo, deben ofrecer un entorno de ejecución protegido donde la máquina virtual pueda correr, o algún tipo de apoyo para el acceso a una red, como puede ser el caso de un API para programar sockets, sobre el cual el protocolo HTTP se pueda implementar. Es el sistema operativo el que ofrecerá acceso al teclado y al posible dispositivo puntero, entregando los correspondientes eventos que surjan. Además, será el encargado de abstraer al MIDP la pantalla, ya que será visto por él como una matriz de pixels, y de ofrecer un interfaz para el acceso al almacenamiento persistente.

Un aspecto muy importante a tener en cuenta es el de la seguridad. CLDC y MIDP no incluyen ningún tipo de prueba en las llamadas al API de las que incluye J2SE, lo que puede suponer un peligro, porque el MIDlet no está limitando en ningún sentido. Es por esto por lo que el usuario debería ser bastante cuidadoso a la hora de instalar nuevas aplicaciones.

Los MIDlets necesitan empaquetarse antes de que sean trasferidos a un dispositivo para su instalación: tanto la subclase MIDlet correspondiente como las clases que requiera y el resto de ficheros necesarios constituirán un único fichero JAR, incluyendo el conocido como manifiesto del JAR (contiene información de empaquetado que indica qué se almacena en el fichero). Adicionalmente se emplea otro segundo fichero conocido como Java Application Descriptor (JAD). El manifiesto del JAR almacena el dispositivo, el nombre y la versión del MIDlet suite en el JAR correspondiente, así como qué ficheros de clase se corresponde con cada MIDlet, información útil para la instalación. El segundo es un fichero de texto que contiene una lista de atributos junto con su valor correspondiente. Algunos atributos están también contenidos en el manifiesto del JAR, ya que éste puede ser grande y su transferencia lenta debido a la baja velocidad del acceso a la red que suelen tener estos dispositivos, en vez de descargar el JAR completo, se descarga el fichero JAD, el cual es mucho más pequeño y rápido de transferir, se muestra por la pantalla del dispositivo y se decide si se instala o no.

2.3.4 Paquetes opcionales en J2ME.

Information Module Profile (IMP), JSR 195: Proporciona un entorno de aplicación para dispositivos embebidos que no tiene grandes capacidades gráficas o con recursos limitados de alguna otra manera: paneles de emergencia, parquímetros, sistemas de alarma domésticos y similares.

Wireless Messaging API (WMA); JSR 120, JSR 205: Proporciona acceso a recursos de comunicación sin cable como la mensajería SMS.

Mobile Media API (MMAPI); JSR 135: Extiende la funcionalidad de la plataforma J2ME incorporando soporte de audio, video y otros tipos de datos multimedia basados en tiempo a dispositivos de recursos limitados.

Location API; JSR-179: Permite la localización de dispositivos móviles para dispositivos con recursos limitados. El API se ha diseñado para generar información sobre la localización geográfica actual del terminal para las aplicaciones Java. El API cubre la obtención de la localización geográfica presente y la orientación del terminal y acceder a una base de datos de mapas almacenados en el terminal.

SIP API; JSR-180: Se utiliza para establecer y gestionar sesiones IP multimedia. El mismo mecanismo se puede utilizar para proporcionar mensajería instantánea, presencia y servicios de juego.

Security and Trust Services API for J2ME; JSR-177: Amplía las características de seguridad para la plataforma J2ME añadiendo APIs de cifrado, servicio de firma digital y gestión de credenciales de usuario.

Mobile 3D Graphics; JSR-184: Permite generar gráficos tridimensionales a frecuencias de imagen interactivas en dispositivos móviles de recursos restringidos.

J2ME Web Services APIs (WSA), JSR 172: Amplía la plataforma de servicios web para incluir J2ME. Estas APIs permiten que los dispositivos J2ME puedan ser clientes de servicios web mediante un modelo de programación consistente con la plataforma estándar de servicios web.

Bluetooth API; JSR-82: Proporciona un estándar para la creación de aplicaciones Bluetooth, de forma que las aplicaciones desarrolladas con el paquete opcional puedan ejecutarse utilizando esta tecnología.

J2ME RMI Optional Package; JSR 66: Permite a dispositivos de consumo y aplicaciones embebidas interactuar como y con aplicaciones distribuidas.

JDBC API; JSR-169: Define un subconjunto del API JDBC 3.0 para procesar datos de repositorios, habitualmente BBDD relacionales, mediante SQL y para manipular datos tabulares como si fueran JavaBeans.

Contactless API; JSR-257: Utilizado para comunicaciones por proximidad o basadas en contacto. Permite tanto el descubrimiento de tarjetas, como el intercambio de datos con éstas, ya sean ndef, rfid o lectores externos.

2.4 Dispositivos móviles.

2.4.1 ¿Qué es un dispositivo?

Antes de entrar detalladamente a describir algunos de los dispositivos móviles, vamos a concretar detalladamente el concepto de dispositivo que estará subyacente en el resto de este apartado. En definitiva:

- son aparatos pequeños,
- con algunas capacidades de procesamiento,
- móviles o no,
- con conexión permanente o intermitente a una red,
- con memoria limitada,
- diseñados específicamente para una función, pero que pueden llevar a cabo otras más generales.
- Normalmente se asocian al uso individual de una persona, tanto en posesión como en operación, el cual puede adaptarlos a su gusto.
- La mayoría de estos aparatos pueden ser transportados en el bolsillo del propietario.
- Otros están integrados dentro de otros mayores, controlando su funcionalidad.

Definimos teléfono dispositivo móvil a aquel dispositivo electrónico de comunicación, normalmente de diseño reducido y sugerente y basado en la tecnología de ondas de radio (es decir, que transmite por radiofrecuencia), y que tiene diferentes funcionalidades. Su rasgo característico principal es que se trata de un dispositivo portable e inalámbrico, esto es, que se puede llevar encima debido a su tamaño y que no requiere de ningún tipo de cableado para llevar a cabo algún tipo de comunicación, ya sea conectarse a otro dispositivo, a Internet o realizar una llamada telefónica.

En este proyecto nos centraremos en los teléfonos móviles, pues la herramienta desarrollada será utilizada por éstos.

Además de ser capaz de realizar llamadas como cualquier otro teléfono convencional, un móvil más o menos moderno suele incorporar un conjunto de funciones adicionales, tales como mensajería instantánea (sms), agenda, juegos, etc., que aumentan la potencialidad de utilización de estos dispositivos.

Es más, su desarrollo y exigencia ha llegado a tal punto, que ya se puede hablar incluso de términos tales como memoria RAM y ligarlos al uso de móviles, dentro información de todo tipo (audio, video, texto, etc.), lo que hace de ellas un complemento perfecto tanto para el hombre de a pie como para el de negocios.

2.4.2 Historia.

La comunicación inalámbrica tiene sus raíces en la invención del radio por Nikola Tesla en los años 1880, aunque formalmente presentado en 1894 por un joven italiano llamado Guglielmo Marconi.

El teléfono móvil se remonta a los inicios de la Segunda Guerra Mundial, donde ya se veía que era necesaria la comunicación a distancia, es por eso que la compañía Motorola creó un equipo llamado Handie Talkie H12-16, que es un equipo que permite

el contacto con las tropas vía ondas de radio que en ese tiempo no superaban más de 600 kHz.

Fue sólo cuestión de tiempo para que las dos tecnologías de Tesla y Marconi se unieran y dieran a la luz la comunicación mediante radio-teléfonos: Martin Cooper, pionero y considerado como el padre de la telefonía celular, fabricó el primer radio teléfono entre 1970 y 1973, en Estados Unidos. Cuenta la leyenda que la primera llamada la realizó desde una calle de Nueva York a Joel Engel, investigador de Bell Labs (que era competencia directa de Motorola) para comunicarle que le estaba llamando desde su teléfono móvil ya funcional.

Ilustración 18. Martin Cooper es considerado como el padre del teléfono móvil.

"La gente desea hablar con la gente - no en una casa, o en una oficina, o en un coche. Dales la opción, y la gente exigirá la libertad para comunicarse dondequiera que este, desencadenándose del infame alambre de cobre. Es esa libertad que intentamos demostrar vividamente en 1973..."[4]

En 1979 aparecieron los primeros sistemas a la venta en Tokio (Japón), fabricados por la Compañía NTT. Los países europeos no se quedaron atrás y en 1981 se introdujo en Escandinavia un sistema similar a AMPS (Advanced Mobile Phone System). Y si bien Europa y Asia dieron los primeros pasos, en Estados Unidos, gracias a que la entidad reguladora de ese país adoptó reglas para la creación de un servicio comercial de telefonía celular, en 1983 se puso en operación el primer sistema comercial en la ciudad de Chicago.

Este fue el inicio de una de las tecnologías que más avances tiene, aunque continúa en la búsqueda de novedades y mejoras.

Resumiendo, hace una década aproximadamente los teléfonos celulares se caracterizaban sólo por llamar, pero ha sido tanta la evolución que ya podemos hablar de equipos multimedia que puede llamar y ejecutar aplicaciones, jugar juegos 3D, ver videos, ver televisión y muchas cosas más. En fin, debemos tener conciencia y prepararnos para lo que se viene más adelante y pensar que el teléfono celular ya no es tan sólo para hablar.

[4] Martin Cooper (1973)

2.4.3 Generaciones de móviles.

La industria de las comunicaciones móviles ha evolucionado en tres etapas, siendo cada generación más fiable y flexible que la anterior:

2.4.3.1 Móviles de primera generación (1G)

Fue analógica y limitada en capacidad de roaming, permitía solamente llamadas de voz con baja calidad y los teléfonos se diseñaron para uso en vehículos. AMPS fue el principal estándar de primera generación y se desarrolló entre 1982 y 1992. El sistema analógico empleado todavía en Europa, el TACS (Total Access Communications System), se basa en AMPS.

2.4.3.2 Móviles de segunda generación (2G)

Son los sistemas actualmente en uso como GSM. Permite la transmisión de voz y datos. Son sistemas digitales basados en conmutación de circuitos que aplican técnicas avanzadas de uso del espectro radioeléctrico y con capacidades de roaming mejoradas. Se basan en un ancho de banda de 9,6 kbps para datos y fax. Significa un incremento en la capacidad de la red, una mejora en la calidad de voz, incorporación de la transmisión de datos, reducción de tarifas y los primeros servicios de valor añadido, como son los mensajes cortos SMS.

2.4.3.3 Móviles de segunda generación y media (2.5G)

Son los sistemas GPRS (General Packet Radio Service) y EDGE (Enhanced Data Rates for GSM Evolution), estos sistemas ofrecen mejoras tecnológicas en las redes 2G actuales tendentes a entregar capacidades 3G; es decir, suponen una mejora en la transmisión de los datos, con una velocidad que puede llegar hasta los 384 kbps, esta tecnología basada en paquetes, ya es adecuada para muchas aplicaciones y además integra WAP, MMS y juegos móviles SMS.

2.4.3.4 Móviles de tercera generación (3G). UMTS

Es un salto enorme sobre los actuales. Pensada para roaming global, transmisión de datos de alta velocidad a través de técnicas avanzadas de conmutación de circuitos y de paquetes, soporta tecnología IP y ATM, lo que posibilita el acceso a Internet, y en general aplicaciones multimedia móviles, con servicios personalizados y basados en la localización de los usuarios. El nuevo modelo de negocio es radicalmente distinto del actual y entran en juego nuevos agentes, como son los proveedores de contenidos y los proveedores de aplicaciones. Esta tecnología 3G está recogida en Europa por el estándar UMTS.

2.4.3.5 Móviles de cuarta generación (4G).

A día de hoy no hay ninguna definición de la 4G, pero podemos resumir en qué consistirá en base a lo ya establecido. Estará basada totalmente en IP siendo un sistema de sistemas y una red de redes, alcanzándose después de la convergencia entre las redes de cables e inalámbricas así como en ordenadores, dispositivos eléctricos y en tecnologías de la información así como con otras convergencias para proveer velocidades de acceso entre 100Mbps en movimiento y 1Gbps en reposo, manteniendo una calidad de servicio de end-to-end de alta seguridad para permitir ofrecer servicios de cualquier clase en cualquier momento, en cualquier lugar, con el mínimo coste posible.

El WWRF (Wireless World Research Forum) define 4G como una red que funcione en la tecnología de Internet, combinándola con otros usos y tecnologías tales como Wi-Fi y WiMAX. La 4G no es una tecnología o estándar definido, sino una colección de tecnologías y protocolos para permitir el máximo rendimiento de procesamiento con la red inalámbrica más barata. El IEEE aún no se ha pronunciado designando a la 4G como “más allá de la 3G”.

En Japón ya se está experimentando con las tecnologías de cuarta generación, estando NTT DoCoMo a la vanguardia. Esta empresa realizó las primeras pruebas con un éxito rotundo (alcanzó 100 Mbps a 200 Km./h) y espera poder lanzar comercialmente los primeros servicios de 4G en el año 2010. En el resto del mundo se espera una implantación sobre el año 2020.

2.4.4 Nokia 6131 NFC.

El teléfono móvil Nokia 6131 NFC cuenta con la tecnología NFC integrada y está especialmente dirigida a desarrolladores de aplicaciones, aunque eso no impide su utilización a otro tipo de usuarios. Las principales características de este teléfono se enumeran a continuación:

Principales características

- Con NFC, sus tarjetas de crédito, tarjetas de fidelidad, y la tarjeta de viaje en un solo lugar.
- Explorar móviles y las noticias con un simple toque a una tarjeta RFID.

El uso de este dispositivo para el pago de billetes y efectos requiere de un servicio de suscripción y la instalación de una aplicación segura en el dispositivo. Para obtener información sobre la capacidad de servicio y la instalación, póngase en contacto con su proveedor de servicios local.

Frecuencia de funcionamiento

- Cuádruple banda GSM / EDGE con capacidad de cobertura en los cinco continentes

Tamaño

- Volumen: 75 cc.
- Peso: 104 g.
- Dimensiones: 92 x 47 x 20 mm.

Pantalla e interfaz de usuario

- Pantalla principal: 2,2 "QVGA TFT con hasta 16,7 millones de colores reales (240 x 320 píxeles)
- Pantalla externa: 1,36" TFT (128 x 160 píxeles)
- Mejora de la interfaz de usuario Serie 40

Imágenes

- Cámara de 1,3 megapíxeles con zoom digital 8x
- Visor de pantalla completa tanto para la pantalla principal como para exterior

Mensajes

- Buzón de entrada común para SMS y MMS.
- MMS 1.2 para la creación, recepción, edición y envío de mensajes multimedia de hasta 300 KB de tamaño.
- Enviar mensajes de correo electrónico con archivos adjuntos (soporta SMTP, POP3, e IMAP4).
- Mensajería instantánea.
- Push to talk (Pulsar para hablar).
- Mensajería de audio Nokia Xpress - enviar mensajes de voz y clips de sonido a través de MMS.

Multimedia

- Reproductor de música con soporte para diferentes formatos.
- Radio estéreo FM
- Streaming de vídeo.
- Tonos de llamada de vídeo.
- Tonos de llamada MIDI polifónicos
- DRM (gestión de derechos digitales) versión 1.0

Funciones de memoria

- 11 MB de memoria de usuario
- Capacidad de memoria expansible con tarjeta de memoria microSD de 2GB.

Aplicaciones

- Java MIDP 2.0 juegos y aplicaciones

Conectividad

- Near field communication (NFC) con capacidad para leer, escribir, y comunicación peer to peer.
- Chip de seguridad integrado que permite almacenar de forma segura su información personal.
- Nokia PC Suite con conectividad USB, Bluetooth, y IrDa.
- Bluetooth versión 2.0 con la especificación Enhanced Data Rate (incluye acceso SIM y perfiles de auricular y manos libres)
- Sincronización de datos SyncML
- Ranura para tarjeta de memoria microSD
- Conector Pop-Port ™ con conectividad USB

Navegación

- Integrated XHTML browser Navegador XHTML integrado
- Smart content download, OMA Digital Rights Management 1.0 Descarga inteligente de contenidos, gestión de derechos digitales OMA 1.0

Transferencia de datos

- EDGE (EGPRS): multislot class 10 EDGE (EGPRS): clase 10 multiintervalo
- GPRS: multislot class 10 GPRS: clase 10 multiintervalo

Gestión de información personal (PIM)

- Administrar su tiempo y la información con el aumento de calendario, y ver sus notas sobre el nuevo modo de espera activa
- Para listas de tareas, notas, calculadora, reloj alarma y temporizador de cuenta atrás (normal y temporizador de intervalo)

Near field communication (NFC)

- De pago sin contacto y la capacidad de venta de entradas
- El acceso a los servicios móviles y la información con un simple toque
- Utiliza la especificación de Java requisito 257 (JSR 257) para el tercero aplicaciones NFC

Funciones de voz

- Push to talk: Seleccione la persona o grupo que desea hablar y pulse la tecla de Pulsar para hablar (una pulsación larga de la tecla de subir volumen) para comunicarse
- Marcación por voz mejorada con SIND (Altavoz independiente nombre de marcación)
- Altavoz manos libres integrado con altavoz de alta calidad para una mejor experiencia de audio
- Comandos de voz
- Grabación de voz

Servicios digitales

- Interfaz de usuario (UI) de temas como fondos de escritorio y salvapantallas animados y tonos de llamada
- Tonos de llamada: vídeo, tonos de llamada MP3, Tonos y verdadera señal MIDI, alerta, tonos y juegos de azar con el apoyo de 64 tonos polifónicos
- Posibilidad de descarga OTA para: Temas, Tonos Cierto, los tonos de llamada MP3, tonos de llamada MIDI, salvapantallas, fondos de escritorio, 3GPP Streaming, imágenes y vídeos, juegos Java Serie 40 y las aplicaciones, garantizar la personalización de chip, el valor o el tiempo de recambio en los billetes de transporte, y móviles servicios

El contenido del paquete de ventas

- Nokia 6131 NFC teléfono
- Nokia Batería BL-4C
- Nokia Cargador compacto AC-3
- Nokia Auricular Estéreo HS-23
- Guía del usuario y guía de inicio rápido

Gestión de la energía

- Batería BL-4C de hasta 3,4 horas de duración en conversación y 240 horas en reposo.

Capítulo 3.

Desarrollo de la aplicación.

3.1 Objetivos del PFC.

Este proyecto surge como continuación de un trabajo dirigido acerca de la tecnología Near Field Communication (NFC). Una vez decidido cual iba a ser el tema principal del proyecto, se procedió a determinar que herramienta se podría desarrollar y en que entorno y escenarios. Surgió así la idea de una herramienta para realizar compras a través del móvil, donde no fuera necesario dinero en metálico sino que tuviéramos una aplicación en el móvil que gestionase directamente nuestro dinero (una especie de monedero electrónico) y que mediante el uso de NFC pudiéramos hacer la compra simplemente acercando el teléfono al producto. Una vez comprados todos los productos, no haría falta más que pasar por caja donde validarían nuestra compra.

Esta fue la idea principal del proyecto, pero tras una reunión posterior decidimos cambiar la aplicación y centrarnos en otra nueva que se nos había ocurrido y que nos resultó más interesante. La nueva idea consistía en desarrollar una herramienta que seria utilizada en reuniones de empresa. En éstas, se establecería un escenario de proyección de contenidos distribuida donde cada usuario llevaría un móvil dotado con la capacidad NFC de tal manera que llevase sus archivos (.doc, .ptt, .pdf, etc) en la memoria del teléfono. Haciendo uso de NFC cada usuario podría enviar un archivo a un nodo coordinador y que este fuese enviado a un PC (mediante Bluetooth) que seria el encargado de presentarlo a través de un proyector. El escenario original seria el que muestra la siguiente figura:

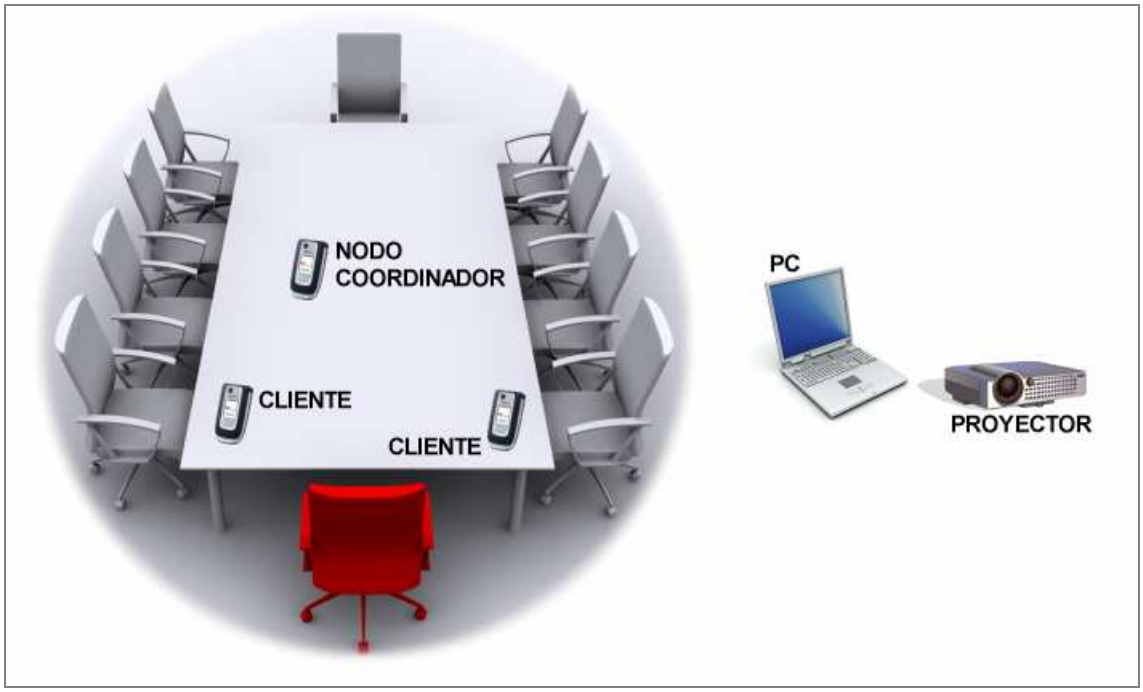

Ilustración 19. Escenario original del proyecto.

Tras estudiar detenidamente las características de este escenario, se decidió a eliminar el nodo coordinador y sustituirlo por una tarjeta NFC que contuviese la dirección Bluetooth del PC directamente. Ya que la transmisión final entre el nodo

coordinador y el PC iba a ser a través de bluetooh (por NFC no se podría debido a la limitación del alcance – seria necesario acercar el teléfono al PC y que este tuviera un lector de NFC para poder recibir el archivo) parecía mejor tener una tarjeta NFC que almacenase la dirección Bluetooth del PC de tal manera que la transmisión de los archivos se hiciera de modo automático. El escenario final es el que muestra la siguiente figura:

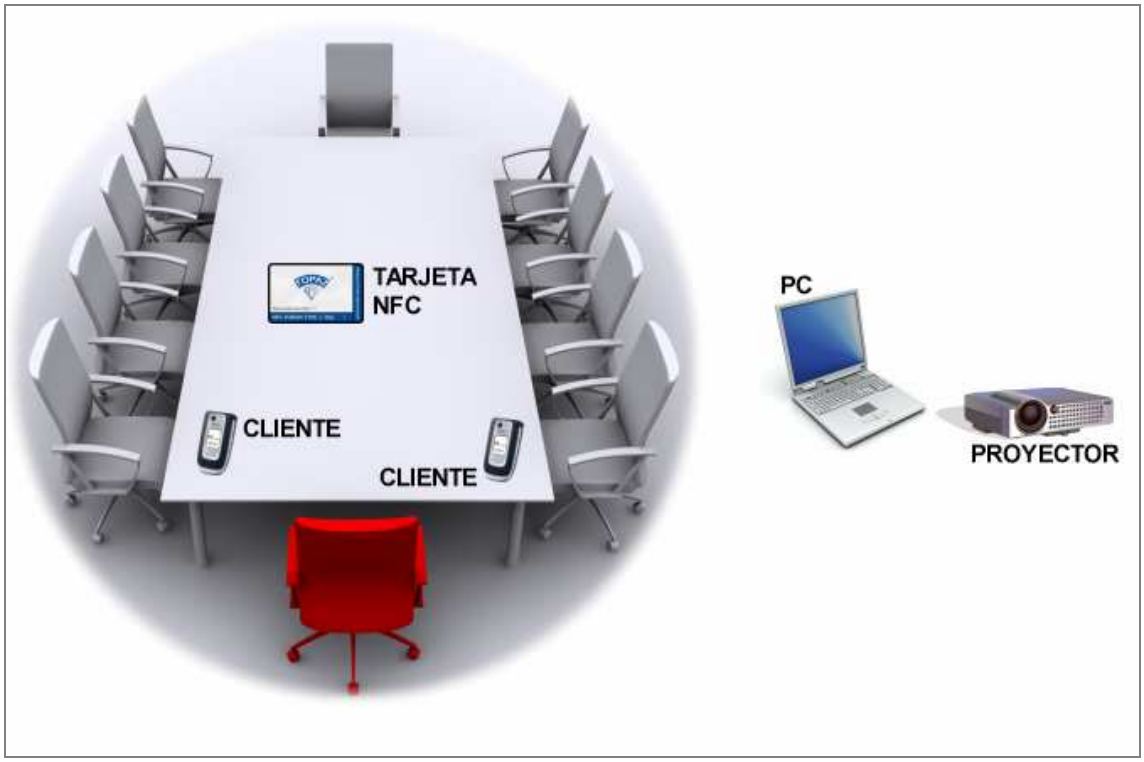

Ilustración 20. Escenario final del proyecto.

Con esta herramienta en este escenario buscábamos una serie de características concretas, que se pueden resumir en lo siguiente:

- Fácil de usar: esta herramienta no requiere de altos conocimientos por parte del usuario final, todos los detalles de bajo nivel son transparentes hacia él, por lo que el uso de la paliación resulta sencillo e intuitivo.

- Dinámica: gracias a esta herramienta las reuniones de empresa resultarían más dinámicas. Todos los usuarios podrían participar mostrando datos para sus exposiciones de manera rápida, simplemente tocar la tarjeta NFC y seleccionar el archivo a enviar. No son necesarias las confirmaciones, ni por parte del usuario que desea enviar ni por parte del PC a la hora de recibir, eliminando axial tiempos innecesarios. Además de esta manera cada usuario estará sentando en su sitio sin tener que levantarse al PC para conectar un pendrive usb con sus archivos.

- Adaptable: esta herramienta permite la participación de un gran número de usuarios, sin que ello afecte a su funcionamiento. No es necesaria ninguna gestión si en un determinado momento un usuario se marcha o si por lo contrario uno nuevo entra.
- Rapidez: gracias a la tecnología Bluetooth, tenemos un sistema de transmisiones rápido.

- Seguridad: debido a que NFC es una tecnología inalámbrica, pero de corto alcance, no existe la posibilidad de acceso a la lectura de la tarjeta NFC por usuarios malintencionados. La transmisión de los archivos por Bluetooth se hace de una determinada manera que hemos determinado (protocolo), por tanto tampoco es posible la transmisión al PC de archivos por parte de usuarios malintencionados. Como ya se vio en el apartado de seguridad de Bluetooth, esta tecnología permite diferentes modos de seguridad así como el cifrado de datos. En nuestro caso, no es necesaria la autenticación por parte del usuario (solo los usuarios de esa empresa podrán tener acceso a este software) y la transmisión de los datos se hace automáticamente sin necesidad de confirmaciones. Esto podría verse como una vulnerabilidad del sistema, pero para un entorno de una reunión de empresa no creemos que fuese necesario un alto nivel de seguridad en ese aspecto.

- Efectividad: La probabilidad de fallo durante el uso de estas tecnologías inalámbricas, como son Bluetooth y NFC, es ciertamente muy baja.

- Eficiencia: Ambas tecnologías permite la transferencia de información de manera muy eficiente desde el punto de vista de consumo de energía.

3.2 Arquitectura de la aplicación.

En el apartado anterior se ha visto el escenario principal de la herramienta de proyección de contenidos distribuida. Ahora, se analizara la arquitectura de la aplicación, módulos de los que consta, herramientas adicionales que se han necesitado, etc.

Toda la herramienta se ha desarrollado en 2 aplicaciones, la que seria la parte del cliente (software que ira instalado en cada uno de los teléfonos móviles de los usuarios) y la parte del servidor (software instalado en el PC). Además, se han programado 2 aplicaciones adicionales para la lectura y escritura de las tarjetas NFC. Existen algunas aplicaciones de ejemplo que nos permiten escribir en este tipo de tarjetas, así como lectores externos que se conectan al PC y nos permiten leer y escribir en éstas, pero en nuestro caso se ha optado por programar las dos aplicaciones desde el principio.

De tal modo, las aplicaciones finales que tenemos son:

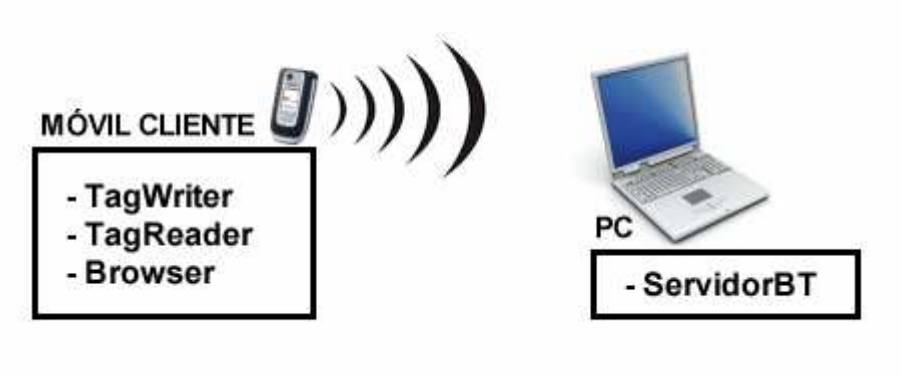

Ilustración 21. Aplicaciones.

Poco a poco se irán explicando cada una de estas aplicaciones, desde las especificaciones iniciales que se pensó que deberían tener, al resultado final, pasando por toda la implementación de cada una de ellas.

3.3 Implementación de utilidades NFC.

En primer lugar se programaron las aplicaciones que harían uso de NFC, y que en resumidas cuentas, no son mas que dos Midlets j2me, uno que escribe en tarjetas y otro que le la información contenida en estás. Como ya se vio en el capitulo del estado del arte, NFC define varios modos de funcionamiento. En este caso, se utilizará el modo pasivo, de tal modo que es el teléfono el que genera un campo magnético lo que permite la transferencia de información entre éste y las tarjetas.

Para la implementación de funcionalidades NFC en j2me, es necesario que nuestro Midlet haga uso del API JSR-257[5]. Este API esta compuesto de 5 paquetes que se resumen en la siguiente tabla:

Paquete java	Interfaces	Clases	Excepciones
javax.microedition.contactless A mandatory package that contains all the target discovery and classes common to all targets	**TagConnection** **TargetListener** **TargetProperties** **TransactionListener**	**DiscoveryManager** **TargetType**	**ContactlessException**
javax.microedition.contactless.ndef An optional package for communicating with NDEF formatted data tags	**NDEFRecordListener** **NDEFTagConnection**	**NDEFMessage** **NDEFRecord** **NDEFRecordType**	
javax.microedition.contactless.rf An optional package for communicating with RFID (no NDEF formatted data) tags	**PlainTagConnection**		
javax.microedition.contactless.sc An optional package for communicating with external smartcards	**ISO14443Connection**		
javax.microedition.contactless.visual An optional package for reading and generating visual tags	**ImageProperties** **VisualTagConnection**	**SymbologyManager**	**VisualTagCodingException**

Ilustración 22. Paquetes del API JSR-257.

Este API permite el descubrimiento de tarjetas y el intercambio de datos con tarjetas, ya sean NDEF, RFID o lectores externos. También incluye soporte para Visual Cards. El siguiente diagrama muestra la relación entre las diferentes clases y paquetes:

[5] Contactless API – utilizada para comunicaciones por proximidad o basadas en contacto.

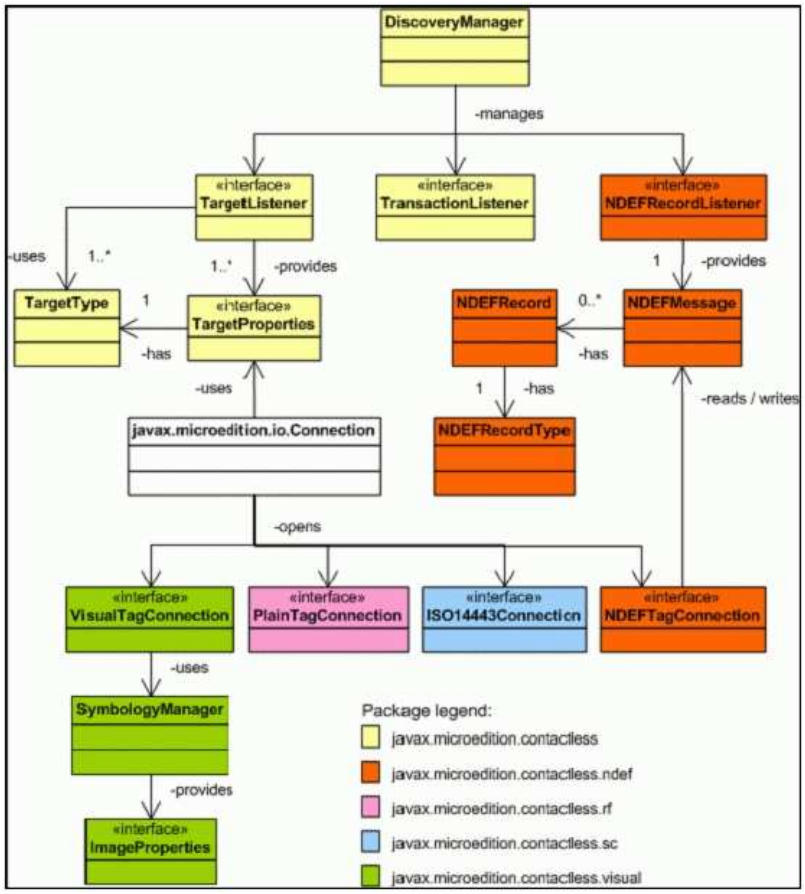

Ilustración 23. Diagrama relacional del API JSR-257.

En nuestro caso solo se hará uso de los dos primeros paquetes. El primero se utilizara para establecer las conexiones y el descubrimiento de las tarjetas, mientras que el segundo se utilizará para la transmisión de información, ya sea para lectura o escritura.

3.3.1 TagWriter

Este Midlet ha sido implementado para poder disponer de la posibilidad de escribir en las tarjetas. Esta aplicación complementaria al proyecto nos permitirá configurar la tarjeta NFC con la dirección Bluetooth del PC donde se vaya a realizar la reunión. Los usuarios participantes no tendrán más que arrancar su software en el móvil y leer el contenido de la tarjeta, de tal modo, ya estarán listos para poder proyectar los contenidos.

Este Midlet es muy sencillo, pues nada mas muestra un elemento Form donde nos indica que escribamos la dirección Bluetooth que queremos grabar en la tarjeta y una vez hecho no hace falta más que acercar el teléfono a la tarjeta para grabar los datos. Aquí podemos ver la vista principal de esta aplicación:

Ilustración 24. Vistas de la aplicación TagWriter.

La programación de este Midlet no fue sencilla, debido en gran parte a la escasa documentación que hay acerca de NFC para j2me. El Midlet debe de implementar la interfaz TargetListener, lo que nos permite que cada vez que se detecte una tarjeta se genere un evento y podamos crear así la conexión y escribir los datos. Es necesario registrar el TargetListener en el DiscoveryManager, que se encargará de llamar al método targetDetected() pasándole un array de TargetProperties. Cada uno de los elementos de ese array, representa una tarjeta detectada. Cada una de esas tarjetas incluye una URL que representa una conexión con el dispositivo móvil. Una vez detectada la tarjeta se procede a abrir una nueva conexión y se escriben la dirección bluetooh que se puso en el formulario.

Dado que la URL del dispositivo indica el protocolo de comunicación, un manejador de ese protocolo que debe estar disponible para comunicarse con el dispositivo recién descubierto. El API JSR-257 no se ocupa del protocolo utilizado, y relega a nivel de protocolo de comunicación los detalles de la conexión. Una vez establecida una conexión, el contenido de esa conexión puede ser recuperado, incluido los *ServiceRecord* almacenados en el dispositivo.

3.3.2 TagReader.

La funcionalidad de este Midlet es la de poder leer las tarjetas NFC, para luego mas tarde integrar ésta en la aplicación que irá instalada en el teléfono. Basta con acercar el teléfono móvil a la tarjeta que deseamos leer, para así poder leer el contenido. Como ya se vió antes, solo contendrá una dirección Bluetooth del PC que ira en la sala de reuniones.

Al igual que el otro Midlet TagWriter, este es muy sencillo mostrando un Form donde nos indica que toquemos la tarjeta que deseamos leer. Una vez leída la tarjeta nos mostrará la dirección Bluetooth que ésta contenía. La vista de la aplicación en el teléfono la podemos ver en la siguiente figura:

Ilustración 25. Vistas de la aplicación TagReader.

El Midlet TagReader implementa la interfaz TargetListener, que como ya vimos nos permite que cada vez que se detecte una tarjeta se genere un evento y podamos crear así la conexión y leer los datos. Es necesario registrar el TargetListener en el DiscoveryManager, que se encargará de llamar al método targetDetected() pasándole un array de TargetProperties.

A continuación se muestra un fragmento de código que muestra como se abre una conexión cuando es detectada una nueva tarjeta.

```
public void targetDetected(TargetProperties[] targetProperties) {
    // exit si no se encuentran tarjetas
    if (targetProperties.length == 0) {
            return;
    }
    // en caso contrario leeremos su uid y los datos
    TargetProperties tmp = targetProperties[0];
        NDEFTagConnection ndefConnection = null;
        try {
                ndefConnection = getNDEFTAGConnection(targetProperties);
                if(ndefConnection == null){ //error al leer la tarjeta
                        return;
                }
                //aqui se procedería a leer la tarjeta o a escribir en ella
                ndefConnection.close();
        } catch (Exception e) {
                e.printStackTrace();
        }
}
```

Una vez implementadas estas dos aplicaciones adicionales, y habiendo pasado un exhaustivo conjunto de pruebas para comprobar su funcionamiento (ver batería de pruebas en el apartado 3.6), se integro la funcionalidad de lectura en la aplicación principal. Recordamos que una vez arrancada la aplicación en el teléfono móvil lo primero que era necesario era la lectura de la tarjeta para disponer de la dirección Bluetooth del PC.

3.4 Implementación file browsing.

JSR 75[6] es una especificación que estandariza el acceso a información que reside en teléfonos móviles y PDAs desde Java. Este API esta compuesto por dos paquetes:

- javax.microedition.pim: nos permite el acceso a los datos PIM (Personal Information Management) tales como contactos, calendario, notas, etc.

- javax.microedition.io.file: permite el acceso a los sistemas de archivos locales del dispositivo, tales como imágenes, canciones, videos, y cualquier fichero almacenado en la memoria del teléfono.

Acceder a este tipo de información se considera una posible vulnerabilidad, por lo que ambos paquetes tienen acceso restringido y es necesario tener permisos para poder utilizarlos. Puede resultar molesto que nuestro Midlet nos pida permiso por medio de una pregunta cada vez que se quiere acceder a los ficheros sistema o la base de datos PIM. Sin embargo, si se firma el Midlet se dispone de una opción de acceso extra cuando se instala en el teléfono móvil. La firma del Midlet se puede hacer a través de lo que se denominan "terceros de confianza", como son Thawte o Verisign que consiste en la emisión de un certificado de tercer nivel por parte de una de estas organizaciones.

Para este proyecto interesa el segundo paquete, FileConnection, que usaremos para acceder a la memoria del teléfono para seleccionar un archivo y así poder enviarlo al PC.

Las operaciones de lectura y escritura en J2ME son manejadas usando GCF (Generic Connection Framework) a través de interfaces Connection. Las diferentes conexiones se construyen utilizando una URL, como pueden ser http://, sockets://, file://, etc. En principio GFC es suficiente para soportar las conexiones para el acceso a ficheros, aunque esto nunca ha sido una parte obligatoria de J2ME o MIDP y en general se ha quedado fuera de la mayoría de las implementaciones. Es importante tener en cuenta las implicaciones con respecto a la seguridad, privacidad y la estabilidad del sistema.

El API JSR 75 asume la existencia de un sistema de archivos en el dispositivo que puede ser localizado, ya sea la memoria del teléfono, memoria flash o memoria externa (como pueden ser las tarjetas). Este API no pretende ser un sustituto para el RMS (Record Management System), si no que es mas bien un complemento que permite interactuar con aplicaciones nativas.

El requerimiento mínimo del API es CLCD 1.0, lo que significa que dispositivos J2ME que ni siquiera tienen interfaz gráfico puedan hacer uso de éste. Sin embargo en nuestro caso lo llevaremos a cabo sobre un dispositivo MIDP 2.0.

3.4.1 Clases e interfaces.

El API de FileConnection nos brinda la oportunidad de acceder al sistema de archivos del dispositivo incluyendo Memory Sticks., y proporcionando un gran numero de funcionalidades. Las principales clases e interfaces de este API son:

- FileConnection: interfaz usada para acceder a directorios ya archivos.

[6] JSR 75 - PDA Optional Packages for the J2ME Platform

- Connection: interfaz que contiene un gran numero de métodos para crear y eliminar archivos y carpetas, listar el contenido, etc.
- FileSystemRegistry: clase que permite listar los diferentes dispositivos motados mediante el método listRoots() (ya sean unidades lógicas o virtuales). Además, proporciona métodos para registrar escuchadores que notifican cambios en el sistema de archivos.
- ConnectionClosedException: excepción que surge cuando se intenta algún tipo de operación estando la conexión cerrada.

3.4.2 Permisos.

Existen dos tipos de permiso definidos en FileConnection:

- javax.microedition.io.Connector.file.read
- javax.microedition.io.Connector.file.write

El primero nos permite acceder a los ficheros en modo de lectura. El segundo nos permite acceder en modo escritura, y además se utiliza también en operaciones de borrado y renombrado. En caso de no incluir estos permisos en la aplicación, se lanzará una SecurityException cuando se vaya a acceder al sistema de archivos.

También es importante tener en cuenta, que todas las operaciones de lectura y escritura deben ser ejecutadas en un hilo independiente para prevenir cualquier tipo de bloqueo o deadlock.

3.4.3 Usando el API. Ejemplos de código.

El procedimiento para crear un FileConnection se lleva a cabo en la misma forma que crea una conexión GFC, donde la única diferencia es la dirección URL. Para crear un FileConnection, es necesario utilizar el método:

```
FileConnection fc = (FileConnection)Connector.open(String URL);
```

En la siguiente tabla podemos ver como abrir un FileConnection según el origen:

Root Value	How to open a FileConnection
CFCard/	FileConnection fc = (FileConnection) Connector.open (“file:///CFCard/”);
SDCard/	FileConnection fc = (FileConnection) Connector.open (“file:///SDCard/”);
MemoryStick/	FileConnection fc = (FileConnection) Connector.open (“file:///MemoryStick/”);
C:/	FileConnection fc = (FileConnection) Connector.open (“file:///C:/”);
/	FileConnection fc = (FileConnection) Connector.open (“file:////”);

Ilustración 26. Modo de abrir un FileConnection.

Los permisos de lectura y escritura se establecen cuando se crea el objeto FileConnection. El fichero al que apunta la dirección URL no tiene por que existir, por lo que abrir y crear una carpeta/archivo se hace del mismo modo. Es importante cerrar el objeto FileConnection cuando se deje de utilizar mediante:

```
fc.close ();
```

Si intentamos hacer algún tipo de operación sobre un archivo que no existe, o abrimos un InputStream o OutputStream saltará una IOException.

Para comprobar si una carpeta o un archivo existe:

```
fc.exists ();
```

Si no existe y queremos crearlo:

```
fc.create ();
```

Para borrar un archivo o un directorio:

```
fc.delete ();
```

Para Mostar el contenido de un directorio:

```
Enumeration e = fc.list();
while (e.hasMoreElements()) {
        System.out.println(((String)e.nextElement()));
}
```

Este método devuelve una enumeración de todos los directorios y archivos que están presentes en el directorio. Directories are denoted with a trailing slash "/" in their returned name. Existe una segunda versión del método de la lista que le permite ordenar los contenidos con un filtro:

```
list(String filter, boolean includeHidden)
```

Para escribir en un archivo:

```
OutputStream os = fc.openOutputStream();
os.write(new String("hola").getBytes());
os.close();
```

O también se puede utilizar el DataOutputStream para escribir datos primitivos :

```
int i = 9999;
DataOutputStream ds = fc. openDataOutputStream();
ds.writeInt(i);
ds.close();
```

Para leer de un archivo usamos su correspondiente InputStream o DataInputStream de FileConnection:

```
byte[] b = new byte[1024];
InputStream is = fc.openInputStream();
is.read(b);
is.close();
```

Hay una serie de métodos en la clase FileConnection que se utilizan para recuperar información sobre el archivo o directorio específico. He aquí algunas de ellas:

`boolean canRead()` - ¿Es el archivo / directorio de lectura?

`boolean canWrite()` - ¿Es el archivo / directorio de escritura?

`long directorySize(boolean includeSubDirs)` - Devuelve el tamaño de todos los archivos en bytes en el directorio. Si includeSubDirs es cierto el tamaño de todos los subdirectorios están también incluidos.

`long fileSize()` - Devuelve el tamaño del fichero en bytes

`long lastModified()` - Devuelve la fecha en que el archivo / directorio fue modificada por última vez.

3.4.4 Vista de la aplicación.

En las siguientes figuras podemos ver el aspecto que tiene la aplicación cliente para recorrer el sistema de ficheros local del dispositivo, para seleccionar el archivo que mas tarde se enviará al servidor mediante Bluetooth:

Ilustración 27. Vista del sistema de ficheros local.

Una vez seleccionado el archivo, se mostrará una pregunta de confirmación de envío, que permitirá aceptar el envío o cancelar para volver a la vista del sistema de ficheros.

3.5 Implementación de utilidades Bluetooth.

A diferencia de lo que ocurre con la implementación de NFC para J2ME y su correspondiente API, acerca de Bluetooth hay disponible una mayor cantidad de información.

Tanto en libros como en Internet se puede acceder a un gran volumen de información acerca de implementación, tipos de conexión, envío de datos, protocolos de transferencia, etc. Esto no quiere decir que la programación del Midlet principal fuese sencilla, todo lo contrario, fue bastante ardua y tediosa mayormente al inicio. Algunos ejemplos resultaron de bastante ayuda, pues hay algunos aspectos que no están del todo claros en la especificación de Bluetooth para j2me.

En una comunicación Bluetooth existe un dispositivo que ofrece un servicio (servidor) y otros dispositivos acceden a él (clientes). En nuestro caso, partiendo del escenario final que se vio en el apartado de objetivos, los teléfonos móviles serán los clientes (enviarán archivos al PC) y el PC será el servidor, esperará a la recepción de archivos por parte de los clientes para proyectarlos.

Dependiendo de qué parte de la comunicación debamos programar deberemos realizar una serie de acciones diferentes. Un cliente Bluetooth deberá realizar las siguientes:

- Búsqueda de dispositivos. La aplicación realizará una búsqueda de los dispositivos Bluetooth a su alcance que estén en modo conectable.

- Búsqueda de servicios. La aplicación realizará una búsqueda de servicios por cada dispositivo.

• Establecimiento de la conexión. Una vez encontrado un dispositivo que ofrece el servicio deseado nos conectaremos a él.

• Comunicación. Ya establecida la conexión podremos leer y escribir en ella.

Por otro lado, el servidor Bluetooth deberá hacer las siguientes operaciones para poder ofrecer un servicio a los clientes:

• Crear una conexión servidora.

• Especificar los atributos de servicio.

• Abrir las conexiones clientes.

Mientras que en los últimos años el hardware Bluetooth ha avanzado mucho, el lo que respecta al software no se había desarrollado muchas aplicaciones hasta recientemente. Con la aparición del API JSR-82 se popularizó en cierto modo la aparición de aplicaciones Bluetooth usando Java. Este API supuso un gran avance para los desarrolladores. En primer lugar porque estandarizo la forma de programar aplicaciones Bluetooth, y en segundo lugar porque esconde la complejidad del bajo nivel al programador de tal manera que esté puede centrarse más en el desarrollo.

El objetivo primordial de ésta especificación era definir un API estándar abierto, no propietario que pudiera ser usado en todos los dispositivos que implementen J2ME. Por consiguiente fue diseñado usando los APIs J2ME y el entorno de trabajo CLDC/MIDP. Además, el API JSR 82 son muy flexibles, ya que permiten trabajar tanto con aplicaciones nativas Bluetooth como con aplicaciones Java Bluetooth.

Algunas de las características que ofrece este API son:

- Descubrimiento de dispositivos y servicios.
- Registro de servicios.
- Establecimiento de conexiones RFCOMM, L2CAP, OBEX.
- Uso de estas conexiones para la transferencia de datos.
- Seguridad en las comunicaciones.

En resumen toda aplicación basada en Bluetooth se puede dividir en cuatro fases, que son:

- Inicialización de la pila.
- Descubrimiento de dispositivos.
- Descubrimiento de servicios.
- Manejo del dispositivo.
- Establecimiento de la conexión y comunicación.

3.5.1 Inicialización de la pila.

El BCC es un conjunto de capacidades que permiten al usuario o al fabricante resolver cualquier tipo de peticiones que puedan generar un conflicto de aplicaciones definiendo directivas en la pila Bluetooth.

Los dispositivos que implementen el API de bluetooth pueden permitir que diferentes aplicaciones se ejecuten concurrentemente. Para que no haya problemas

entre ambas, de tal manera que una aplicación pueda perjudicar a otra, tenemos el BCC (Bluetooth Control Center).

El BCC puede ser tanto una aplicación nativa, otra aplicación de otro API independiente, o un conjunto de parámetros no modificables definidos por el fabricante. Hay que destacar que el BCC es un elemento muy importante en la arquitectura de seguridad. Se encargará de manejar el modo de seguridad que la pila debe usar y mantendrá las listas de dispositivos seguros.

La pila Bluetooth es la responsable de controlar el dispositivo Bluetooth, por lo que es necesario inicializarla antes de hacer cualquier otra cosa. El proceso de inicialización consiste en un número de pasos cuyo propósito es dejar el dispositivo listo para la comunicación inalámbrica.

Desafortunadamente, la especificación deja la implementación del BCC a los vendedores, y cada vendedor maneja la inicialización de una manera diferente. En un dispositivo puede haber una aplicación con un interfaz GUI, y en otra puede ser una serie de configuraciones que no pueden ser cambiados por el usuario. Un ejemplo sería:

```
...
// Configuramos el puerto
BCC.setPortNumber(“COM1”);
// Configuramos la velocidad de la conexión
BCC.setBausRate(50000);
//Configuramos el modo conectable
BCC.setConnectable(true);
//Configuramos el modo discovery a LIAC4
BCC.setDiscoverable(DiscoveryAgent.LIAC);
...
```

3.5.2 Descubrimiento de dispositivos.

El punto de partida es la clase LocalDevice que representa el dispositivo en el que se esta ejecutando la aplicación. Este objeto se obtiene mediante LocalDevice.getLocalDevice() y permite obtener información sobre el dispositivo: modo de conectividad, dirección Bluetooth y nombre del dispositivo.

Cualquier aplicación basada en Bluetooth, debe realizar una búsqueda de los dispositivos cercanos una vez inicializada la pila. Para ello se debe obtener un objeto DiscoveryAgent que será único y se obtiene a través del objeto LocalDevice.

```
DiscoveryAgent da =LocalDevice.getLocalDevice().getDiscoveryAgent();
```

El objeto DiscoveryAgent nos va a permitir realizar y cancelar búsquedas de dispositivos y de servicios. Y también nos servirá para obtener listas de dispositivos ya conocidos. Los dos tipos de búsquedas que se pueden realizar son:

- Bloqueante: a través del método retrieveDevices(), que devuelve una lista de dispositivos que fueron encontrados en una búsqueda previa o que ya conoce por defecto. Cada uno de los elementos de la lista pertenece a la clase RemoteDevice y representa a un dispositivo Bluetooth con el que se puede comunicar.

- No bloqueante: a través del método startInquiry(), que permite notificar a la aplicación cada vez que se descubre un nuevo dispositivo, por lo que es necesario que ésta tenga registrado un escuchador (listener).

En la siguiente figura se puede ver el diagrama de estados en el proceso de búsqueda de dispositivos:

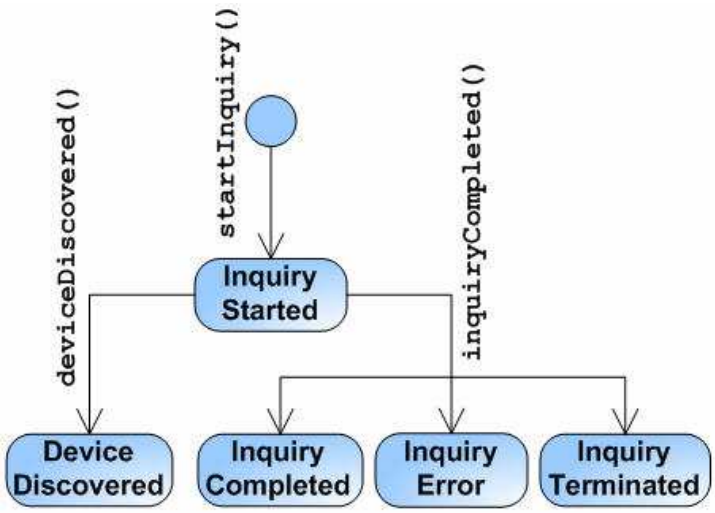

Ilustración 28. Diagrama de estados.

3.5.2.1 Clases e interfaces.

interface javax.bluetooth.DiscoveryListener.

Esta interfaz, que será implementada a conveniencia según la aplicación, se usará para que el dispositivo notifique eventos a la aplicación cada vez que se descubre un dispositivo, un servicio, o se finaliza una búsqueda. Los dos métodos para buscar dispositivos de esta interfaz son:

- deviceDiscovered() – cada vez que un dispositivo se descubre se llama a este método pasándole un objeto RemoteDevice.

- inquiryCompleted() – este método es llamado cuando la búsqueda de dispositivos ha concluido y recibe como parámetro un entero que le indica el motivo de la finalización (con valores INQUIRY_COMPLETED, INQUIRY_ERROR o INQUIRY_TERMINATED).

Además de estos dos métodos, la interfaz cuenta con otro dos métodos que son servicesDiscovered() y serviceSearchCompleted() que se utilizan a la hora de buscar servicios en un determinado dispositivo. Éstos se verán mas adelante en el apartado de descubrimiento de servicios.

interface javax.bluetooth.DiscoveryAgent.

Esta interfaz provee métodos para descubrir dispositivos y servicios. Para descubrir dispositivos, esta clase provee del método startInquiry() para poner al dispositivo en modo de búsqueda, y el método retrieveDevices() para obtener la información de dispositivos previamente encontrados. Además provee del método cancelInquiry() que permite cancelar una operación de búsqueda.

3.5.3 Descubrimiento de servicios.

El siguiente paso después de encontrar los dispositivos cercanos, es el de la búsqueda de servicios. La clase DiscoveryAgent provee de métodos para buscar servicios en un dispositivo servidor Bluetooth e iniciar transacciones entre el dispositivo y el servicio. Este API no da soporte para buscar servicios que estén ubicados en el propio dispositivo.

Para ello es importante comprender primero algunos conceptos. Una aplicación cliente es una aplicación que requiere un servidor para que le ofrezca un servicio. Este servicio puede ser: un servicio de impresión, un servicio de videoconferencia, un servicio de transferencia de archivos, etc. En una comunicación TCP-IP un cliente se conecta directamente a un servidor del que conoce el servicio que ofrece, es decir, conocemos a priori la localización del servidor y el servicio que nos ofrecerá; sin embargo un cliente Bluetooth no conoce de antemano qué dispositivos tiene a su alcance ni cuáles de ellos pueden ofrecerle el servicio que necesita. De modo que un cliente Bluetooth necesita primero buscar los dispositivos que tiene a su alcance y posteriormente les preguntará si ofrecen el servicio en el que está interesado.

Cada servicio es identificado numéricamente, es decir, a cada servicio le asignamos un número, de tal manera que una vez asignado para referirnos al servicio lo haremos a través de este numero asociado. Este identificador se denomina UUID (Universal Unique IDentifier). Además, cada servicio tiene ciertos atributos que lo describen, que al igual que el servicio tendrán un número asociado.

Las búsquedas de servicios, al igual que con los dispositivos, también se realizan mediante el objeto DiscoveryAgent. Concretamente se usa el método searchServices() al que hay que pasarle un objeto DiscoveryListener que recibirá los eventos de la búsqueda, el dispositivo en el que realizar la búsqueda (un objeto RemoteDevice que normalmente se obtiene en la búsqueda de dispositivos), los servicios en los que se está interesado, y los atributos que queremos conocer sobre dichos servicios.

Si se encuentra algún servicio se le notificará a la aplicación a través del objeto DiscoveryListener mediante el método servicesDiscovered(). Éste método recibe un array de objetos ServiceRecord que encapsulan los atributos de servicio que se han solicitado en la búsqueda. Los valores de estos atributos de servicio son objetos DataElement (encapsula los tipos de datos en los que puede ser representado un atributo de servicio, que pueden ser: números enteros de diferente longitud con o sin signo, cadenas de texto, URLs, booleanos, o colecciones de DataElements). Un ServiceRecord es como una tabla que relaciona los identificadores de los atributos con sus valores (objetos DataElement). Cuando finalice la búsqueda de servicios se le notificará a la aplicación mediante una llamada al método serviceSearchCompleted() de la interfaz DiscoveryListener, que recibe como argumento un entero indicando el motivo de la finalización. Este entero puede valer SERVICE_SEARCH_COMPLETED, SERVICE_SEARCH_TERMINATED, SERVICE_SEARCH_NO_RECORDS, SERVICE_SEARCH_ERROR, SERVICE_SEARCH_DEVICE_NOT_REACHABLE.

3.5.3.1 Clases e interfaces.

class javax.bluetooth.DiscoveryAgent

Esta clase provee métodos para descubrir servicios y dispositivos.

interface javax.bluetooth.DiscoveryListener

Este interfaz permite a una aplicación especificar un listener que responda a un evento del tipo Service Discovery o Device Discovery. Cuando un nuevo servicio es descubierto, se llama al método servicesDiscovered(), y cuando la transacción ha sido completada o cancelada se llama a serviceSearchCompleted().

class javax.bluetooth.UUID

Esta clase encapsula enteros sin signo que pueden ser de 16, 32 ó 128 bits de longitud. Estos enteros se usan como un identificador universal cuyo valor representa un atributo del servicio. Sólo los atributos de un servicio representados con UUID están representados en el Bluetooth SDP (Service Discovery Profile).

class javax.bluetooth.DataElement

Esta clase contiene varios tipos de datos que un atributo de servicio Bluetooth puede usar. Algunos de estos son:

- String
- boolean
- UUID
- Enteros con signo y sin signo, de 1, 2, 4 o 6 bytes de longitud
- Secuencias de cualquiera de los tipos anteriores.

Esta clase además presenta una interfaz que permite construir y recuperar valores de un atributo de servicio.

interface javax.bluetooth.ServiceRecord

Este interfaz define el Service Record de Bluetooth, que contiene los pares (atributo ID, valor). El atributo ID es un entero sin signo de 16 bits, y valor es de tipo DataElement. Además, este interfaz tiene un método populateRecord() para recuperar los atributos de servicio deseados (pasando como parámetro al método el ID del atributo deseado).

3.5.4 Registro de servicios.

Cualquier aplicación que ofrezca un servicio Bluetooth debe de cumplir una serie de responsabilidades que se enumeran a continuación:

- Crear un ServiceRecord que describa el servicio ofrecido por la aplicación.
- Añadir el ServiceRecord al SDDB[7] del servidor para avisar a los clientes potenciales de este servicio.
- Registrar las medidas de seguridad Bluetooth asociadas a un servicio.
- Aceptar conexiones de clientes que requieran el servicio ofrecido por la aplicación.
- Actualizar el ServiceRecord en el SDDB del servidor si las características del servicio cambian.
- Quitar o deshabilitar el ServiceRecord en el SDDB del servidor cuando el servicio no está disponible.

3.5.4.1 Responsabilidades del registro de servicio.

El registro de servicio tiene que cumplir una serie de responsabilidades que se pueden ver en el siguiente gráfico.

[7] Service Discovery Database.

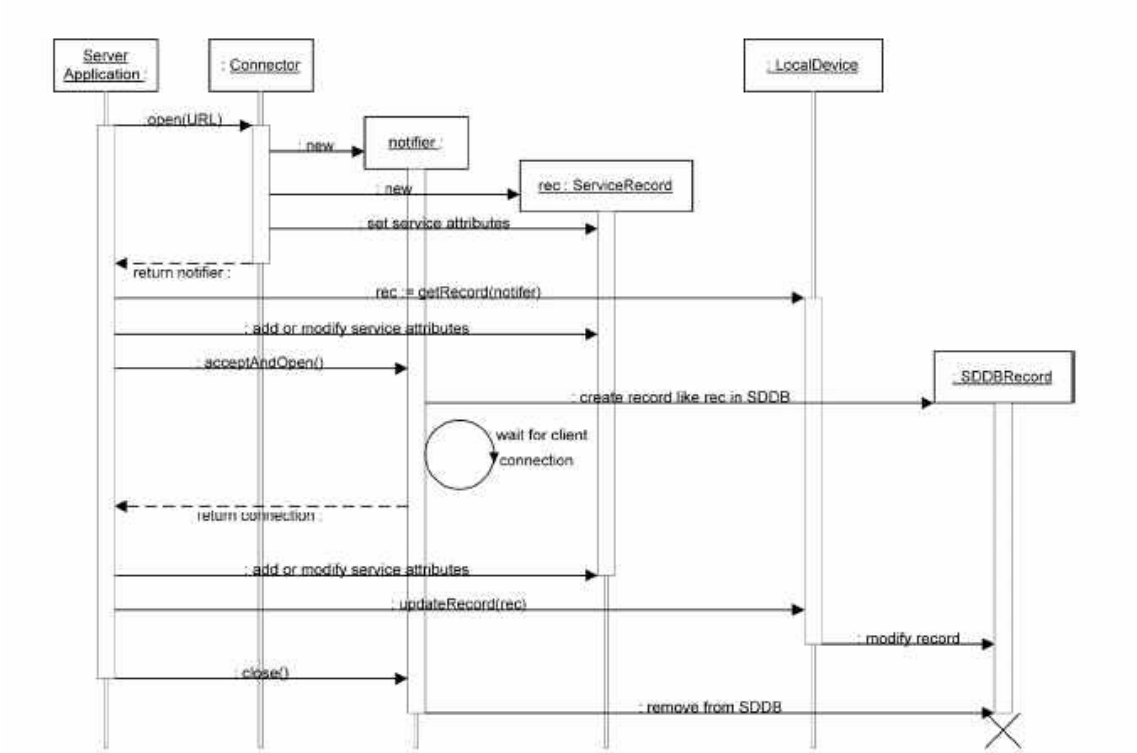

Ilustración 29. Colaboración entre la implementación y la aplicación para el registro del servicio.

Se puede ver que cuando la aplicación llama a Connector.open() con un String conexión URL, la implementación crea un ServiceRecord. El correspondiente registro del servicio es añadido a la SDDB cuando la aplicación servidora llama a acceptAndOpen(). La aplicación servidora puede acceder a dicho ServiceRecord llamando a getRecord() y hacer las modificaciones pertinentes. Las modificaciones se hacen también en el ServiceRecord de la SDDB cuando la aplicación llama a updateRecord(). Finalmente el ServiceRecord es eliminado de la SDDB cuando la aplicación servidora manda un close al notifier.

3.5.4.2 Modos de operación.

- Modo conectable: un dispositivo en este modo escucha periódicamente intentos de iniciar una conexión de un dispositivo remoto.
- Modo no-conectable: un dispositivo en este modo no escucha intentos de iniciar una conexión de un dispositivo remoto.

Para el correcto funcionamiento de una aplicación servidora, es necesario que el dispositivo esté configurado en modo conectable. Es por esto, que en la implementación de acceptAndOpen(), ésta debe asegurarse que el dispositivo local esté en modo conectable (dado que depende del usuario el tener o no el dispositivo en un modo u otro). La implementación hace una petición al BCC para hacer al dispositivo local conectable, si ésta no es posible, se lanzará una excepción BluetoothStateException.

Cuando todos los servicios en la SDDB han sido eliminados o deshabilitados, la implementación puede decidir opcionalmente pedir al dispositivo servidor que pase al modo no-conectable.

Aunque un dispositivo esté en el modo no-conectable (no responde a intentos de conexión), puede iniciar un intento de conexión. Por esto, un dispositivo en modo

no-conectable puede ser un cliente, pero no un servidor. Por lo tanto la implementación no necesita pedir al dispositivo que se ponga en modo conectable si no tiene ningún ServiceRecord en su SDDB.

3.5.4.3. Clases e interfaces.

interfaz javax.bluetooth.ServiceRecord

Un ServiceRecord describe un servicio Bluetooth a los clientes. Está compuesto de unos "cuantos" atributos de servicio. El SDP del servidor mantiene una base de datos de los ServiceRecords. Un servicio *run-before-connect* (la aplicación se ejecuta sin esperar a que haya una conexión establecida) añade su ServiceRecord a la SDDB llamando a acceptAndOpen(). El ServiceRecord provee suficiente información a un cliente SDP para poder conectarse al servicio Bluetooth del dispositivo servidor.

La aplicación servidora puede usar el método setDeviceClasses() para activar alguno de los bits de la clase servidora para reflejar la incorporación de un nuevo servicio.

class javax.bluetooth.LocalDevice

Esta clase provee un método getRecord() que la aplicación servidora puede usar para obtener el ServiceRecord. Una vez modificado, puede ser puesto en la SDDB llamando al método notifier.acceptAndOpen() o updateRecord() de LocalDevice.

class javax.bluetooth.ServiceRegistrationException extends java.io.IOException

La excepción ServiceRegistrationException se lanza cuando se intenta añadir o modificar un ServiceRecord en la SDDB y hay algún error. Estos errores pueden ocurrir:

- Durante la ejecución de Connector.open().
- Cuando un servicio *run-before-connect* invoca a acceptAndOpen() y la implementación intenta añadir el ServiceRecord asociado al *notifier* en la SDDB.
- Después de la creación del ServiceRecord, cuando la aplicación servidora intenta modificar el ServiceRecord en la SDDB usando updateRecord().

3.5.4 Manejo del dispositivo.

Los dispositivos inalámbricos, son más vulnerables a ataques del tipo spoofing y eavesdropping que los demás dispositivos. Es por ello, que la tecnología Bluetooth incluye una serie de medidas para evitar estas vulnerabilidades, como es por ejemplo el salto de frecuencia además de otros mecanismos opcionales como son la encriptación y autenticación, como ya se vió en el apartado 2.2.5 (seguridad en Bluetooth).

Vamos a ver a continuación, las diferentes maneras en las que un dispositivo local responde a un dispositivo remoto. También veremos como usar estos mecanismos de seguridad.

3.5.4.1 Seguridad.

Tanto el cliente como el servidor pueden incluir opcionalmente parámetros al connection string de Connector.open() para especificar la seguridad requerida para la conexión.

Las aplicaciones servidoras usan uno de los métodos open de la clase Connector para crear un objeto notifier que puede ser usado para esperar a que un cliente se conecte. Los parámetros normales del connection string son suficientes para crear un objeto notifier y el apropiado ServiceRecord, sin embargo, añadiendo ciertos parámetros, se pueden requerir autenticación, encriptación, autorización y cambios en el rol maestro/esclavo.

Peticiones de autenticación por parte del servidor:

La autenticación consiste en verificar la identidad del dispositivo remoto. La autenticación implica un reto que se lanza entre los dispositivos, que requiere una clave compartida de 128 bits derivada de un PIN compartido por ambos dispositivos. Si el PIN en ambos dispositivos falla, falla el proceso de autenticación.

El parámetro authenticate tiene la siguiente interpretación cuando se usa en el connection string de una aplicación servidora:

- Si authenticate=true, la implementación intenta identificar a cada cliente que intente conectarse al servicio.
- Si authenticate=false, la implementación no intenta identificar a cada cliente que intente conectarse al servicio.
- Si el parámetro authenticate no está en el connection string, por defecto está a false.

No todos los dispositivos Bluetooth soportan autenticación. Si éste es el caso, y authenticate=true se lanza una excepción BluetoothConnectionException. En algunos casos, puede haber conflictos entre las necesidades de seguridad de una aplicación y las medidas de seguridad tomadas por el dispositivo; algunas implementaciones de la BCC intentan evitar el conflicto preguntando al usuario si quiere cambiar la configuración del dispositivo.

Peticiones de encriptación por parte del servidor:

Cuando está activada, todos los datos transmitidos en ambas direcciones se encriptan. El parámetro encrypt tiene la siguiente interpretación cuando se usa en el connection string de una aplicación servidora:

- Si encrypt=true, la implementación encripta todas las comunicaciones de éste y hacia este servicio.
- Si encrypt=false, no se usa encriptación, pero ésta puede ser requerida por el cliente.
- Si encrypt no está presente en el connection string, por defecto está a false.

Dado que la encriptación requiere una clave compartida, esto significa que la encriptación requiere autenticación. Es decir, sólo ciertas combinaciones de estos parámetros están permitidas. En el caso authenticate=false y encrypt=true provocará una BluetoothConnectionException. Si encrypt=true y el parámetro authenticate no está en el connection string, se considerará que su valor es true.

Al igual que en la autenticación, no todos los dispositivos soportan encriptación. Si encrypt=true y la encriptación no está soportada, se lanzará una BluetoothConnectionException desde el Connector.open().

Peticiones de autorización por parte del servidor:

La autorización Bluetooth es un procedimiento por el cual un usuario de un dispositivo servidor garantiza el acceso a un servicio específico a un cliente específico. La implementación de la autorización puede implicar preguntar al usuario del dispositivo servidor si el dispositivo cliente está autorizado a acceder a dicho servicio.

El parámetro authorize tiene la siguiente interpretación cuando se usa en el connection string de una aplicación servidora:

- Si authorize=true, la implementación consulta con la BCC para determinar si la petición de conexión del cliente se autoriza o no.
- Si authorize=false, todos los clientes tienen acceso al servicio.
- Si no está presente en el connection string, por defecto está a false.

Como en la encriptación, la autorización implica que la identidad del dispositivo cliente debe ser verificada mediante autenticación. Por esto, sólo ciertas combinaciones son válidas. Si authenticate=false y authorize=true se lanzará una BluetoothConnectionException. Si authorize=true y el parámetro authenticate no está presente, se considera que es true.

Al igual que en la autenticación y encriptación, no todos los dispositivos soportan autorización. Si authorize=true y la autorización no está soportada, se lanzará una BluetoothConnectionException desde el Connector.open().

Peticiones de cambio de rol maestro/esclavo por parte del servidor:

Cada red Bluetooth tiene un dispositivo maestro cuya secuencia de salto de frecuencia (frecuency hopping) se usa para sincronizar de uno a siete esclavos. El dispositivo que inicia la formación del canal de comunicaciones es el maestro. A continuación vamos a ver como un esclavo pide el cambio de rol maestro/esclavo.

El parámetro master tiene la siguiente interpretación cuando se usa en el connection string de una aplicación servidora:

- Si master=true, tan pronto como la conexión esté establecida, la implementación hace una petición de que el cliente y el servidor cambien sus papeles.
- Si master=false, el servidor está dispuesto a ser tanto maestro como esclavo.
- Si el parámetro no está definido, por defecto esta a false.

No todos los dispositivos soportan cambios de rol. Si master=true y el cambio de rol no está soportado, se lanzará una BluetoothConnectionException desde el Connector.open().

Peticiones por parte del cliente:

Las aplicaciones cliente también pueden usar los parámetros authenticate, encrypt y master en el connection string. Estos tienen los siguientes significados:

- Si authenticate=true la implementación intenta verificar la identidad del servicio.
- Si encrypt=true, la implementación encripta todas las comunicaciones, encrypt=true implica que authenticate=true.
- Si master=true, el cliente debe jugar el papel de maestro en la comunicación con el servidor.

Con este API, el único dispositivo que necesita garantizar permiso para usar un servicio es el que ofrece dicho servicio. Por eso, el parámetro authorize no está permitido en conexiones del cliente. Si aparece el parámetro authorize en el connection string del cliente se lanzara una BluetoothConnectionException.

Muchas de las diferentes combinaciones del connection string del cliente y servidor son válidas. La única excepción que no se puede dar es la de que ambos tengan master=true. En este caso el intento de conexión falla. El Connector.open() del cliente lanza una BluetoothConnectionException, mientras que el servidor no tiene conocimiento de este fallo ya que la implementación simplemente rechaza la conexión.

3.5.5 Comunicación.

Una vez encontrado el servicio en el que se está interesado en un determinado dispositivo, el siguiente paso sería establecer la comunicación con el dispositivo servidor y posteriormente la transferencia de datos o archivos. Para ello es muy importante que tanto la aplicación cliente como la aplicación servidora usen el mismo protocolo de comunicación.

El Generic Connection Framework (GFC) del CLDC provee la conexión base para la implementación de protocolos de comunicación. CLDC provee de los siguientes métodos para abrir una conexión:

```
Connection Connector.open(String name);
Connection Connector.open(String name, int mode);
Connection Connector.open(String name, int mode, boolean timeouts);
```

3.5.5.1 Perfil del puerto serie (SPP).

El protocolo RFCOMM provee múltiples emulaciones de puertos serie RS-232 entre dos dispositivos Bluetooth, donde las direcciones de los dos dispositivos definen una sesión. Una sesión puede tener una o más conexiones que dependerá de la implementación en el dispositivo. Además, un dispositivo podrá tener más de una sesión RFCOMM en tanto que esté conectado a más de un dispositivo.

Una aplicación que ofrezca un servicio basado en el perfil de puerto serie es un servidor SPP, mientras que una aplicación que inicie una conexión a un servicio SPP es un cliente SPP. Cliente y servidor residen en los extremos de una sesión RFCOMM.

El servidor SPP debe inicializar el serivicio que vaya a ofrecer y registralo en el SDDB. Un servicio de puerto serie viene representado por un par de objetos emparentados:

- Un objeto que implementa el interfaz StreamConnectorNotifier. Este objeto escucha conexiones clientes que demanden este servicio.

- Un objeto que implemente el interfaz ServiceRecord. Este objeto describe el servicio y como puede ser accedido por dispositivos remotos.

Para crear estos objetos la aplicación servidora usa el método open() de la clase Connector pasandol como argumento la URL de conexión, del siguiente modo:

```
StreamConnectionNotifier server = (StreamConnectionNotifier)Connector.open
("btspp://localhost: F0E0D0C0B0A000908070605040302010;name=SPPservicio");
```

La implementación de Connector.open() además crea un nuevo registro de servicio (ServiceRecord) que representa el servicio SPP. Una implementación de un SPP debe realizar los siguientes pasos cuando crea el registro de servicio.

1. Crear un identificador de un canal servidor RFCOMM, chanN y asignarlo.

2. chanN es añadido al ProtocolDescritorList en el registro de servicio.
3. El UUID (F0E0D0C0B0A000908070605040302010), usado para describir el tipo de servicio ofrecido, es añadido al ServiceClassIDList.

4. El atributo ServiceName es añadido al registro de servicio con el valor “SPPservicio”.

Establecimiento de la conexión en el servidor.

Como se ha visto en el apartado anterior, un servidor crea un objeto StreamConnectionNotifier haciendo:

```
StreamConnectionNotifier server = (StreamConnectionNotifier)Connector.open
("btspp://localhost: F0E0D0C0B0A00090807060504030201O;name=SPPservicio");
```

Un servicio SPP puede aceptar múltiples conexiones de diferentes clientes mediante el uso de acceptAndOpen() repetidamente. Cada cliente accede al mismo registro de servicio y se conecta al servicio usando el mismo canal servidor RFCOMM. En caso de que el sistema no soportase múltiples conexiones, el método acceptAndOpen() lanzaría una BluetoothStateException.

El método close() se usará para cerrar la conexión.Cuando un servicio manda un mensaje close() al StreamConnectionNotifier, el registro de servicio asociado se vuelve inaccesible a los clientes que estén usando el servicio. La implementación debe eliminar el registro de servicio de la SDDB.

Establecimiento de la conexión del cliente.

Para que un cliente pueda establecer una conexión con el servidor, es necesario que previamente haya descubierto el servicio. Una conexión URL del cliente incluye la dirección Bluetooth del dispositivo servidor y el identificador de canal del servidor. El método **getConnectionURL()** en el interfaz ServiceRecord se usa para obtener la conexión URL del cliente.

Invocando el método **Connector.open()** con una conexión URL del cliente, devuelve un objeto StreamConnection que representa la conexión SPP del lado del cliente.

```
ServiceRecord service = ...
String url = service.getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOENCRYPT, false);
StreamConnection connection = null;
StreamConnection con = (StreamConnection) Connector.open (url);
```

3.5.5.2. L2CAP.

El protocolo L2CAP soporta dos tipos de conexiones, orientadas a conexión (bidireccionales) y no orientadas a conexión (unidireccionales). Pero el API que se utiliza soporta sólo canales L2CAP orientados a conexión.

El L2CAPConnectionNotifier le indica a un servidor L2CAP cuando un cliente inicia una conexión. Una vez que la conexión está establecida, se devuelve un objeto L2CAPConnection. El interfaz L2CAPConnection y L2CAPConnectionNotifier extienden el interfaz Connection. Este interfaz puede ser usado para enviar o recibir datos de un dispositivo remoto usando el protocolo L2CAP.

Las conexiones deben de ser configuradas mediante algunos parámetros del canal que son: MTU (Unidad Máxima de Transferencia), tiempo de descarte y calidad de servicio (QoS).

El pseudocódigo para abrir una conexión cliente L2CAP es:

```
try{
L2CAPConnection client = (L2CAPConnection)
Connctor.open(“btl2cap://...:...;ReceiveMTU=xxx;TransmitMTU=yyy”);
}catch(...)
```

El pseudocódigo para abrir una conexión servidor L2CAP es:

```
try{
L2CAPConnectionNotifier server = (L2CAPConnectionNotifier)
Connctor.open(“btl2cap://localhost:...;name=L2CAPEx”);
L2CAPConnection con = (L2CAPConnection) server.acceptAndOpen();
}catch(...)
```

En el caso de que haya habido algún fallo durante la conexión se puede obtener el motivo del fallo usando el método getStatus().

3.5.5.3 OBEX (Protocolo de intercambio de objetos).

Este protocolo no esta definido en el API de Bluetooth pero aun asi se explicara un poco su funcionamiento y utilidad. Como ya se vió en el apartado 2.2.4.3 (pila de protocolos), OBEX es un protocolo diseñado por el IrDA para intercambiar objetos entre clientes y servidor, mediante el establecimiento de una sesión.

En vez de incluir esta funcionalidad en el API Bluetooth, se ha optado por extender el API OBEX y dar soporte a Bluetooth. En nuestro caso, solo nos interesa el soporte Bluetooth.

OBEX implementa la transferencia de objetos estableciendo una sesión, similar al protocolo HTTP. Para iniciarla se usa una petición CONNECT y para terminar ésta una petición DISCONECT. Entre estas dos peticiones, el cliente puede traer objetos del servidor mediante GET, o enviarlos mediante PUT. Los objetos pueden ser archivos, arrays de bytes, etc. El cliente puede además cambiar el archivo o carpeta en uso mediante la petición SETPATH. Otras operaciones permitidas son; ABORT, CREATE-EMPTY, PUT-DELETE. OBEX, como HTTP, tiene métodos que le permiten pasar información adicional entre el cliente y el servidor mediante el uso de cabeceras. Además, OBEX permite la autenticación entre cliente y servidor.

Conexión del cliente.

Para crear una conexión OBEX, el cliente le debe parar el string apropiado al Connector.open(), y este devolverá un objeto javax.obex.ClientSession. Para establecer la conexión OBEX el cliente crea un objeto javax.obex.HeaderSet usando el método createHeaderSet() del interfaz ClientSession. Finalmente el cliente facilita el objeto HeaderSet al método connect() de la interfaz ClientSession.

Para determinar si la petición ha tenido éxito o no, se usa el método getResponseCode() del interfaz HeaderSet, que devuelve un código de respuesta mandado por el servidor, que viene definido en la clase javax.obex.ResponseCodes.

Para la petición DISCONECT, se procede del mismo modo, excepto que en vez de usar el método connect() se usa el método disconnect().

Para completar una operación SETPATH, el cliente llama al método setPath() en el objeto ClientSession. Para especificar el nombre del directorio destino, pone el nombre llamando al método setHeader() del HeaderSet. Si la cabecera es muy larga se lanzará una excepción java.io.IOException.

Para completar una operación GET o PUT, el cliente crea un objeto javax.obex.HeaderSet con el método createHeaderSet(). Después de establecer los valores de cabecera, el cliente llama a los métodos put() o get() del objeto javax.obex.ClientSession.

Para abortar un PUT o un GET, el cliente llama al método abort() del objeto javax.obex.Operation. El método abort() llama además al método close() del objeto Operation.

Conexión del servidor.

Para crear una conexión servidora, el servidor invoca a Connector.open(), que le devuelve un objeto javax.obex.SessionNotifier. Este objeto espera a que el cliente cree una capa de transporte llamando a acceptAndOpen().

El servidor debe crear una nueva clase que extienda la clase javax.obex.serverRequestHandler e implementar aquellos métodos de OBEX a los que da soporte. Las aplicaciones servidoras no deben llamar al método abort(), ya que si no el argumento Operation, que es parte de los métodos onGet() y onPut(), lanzará una java.io.IoException.

3.5.6 Ejemplo de aplicación, “HolaMundo”.

SPPServidorMidlet

```
package SPPBluetooth;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class SPPServidorMIDlet extends MIDlet implements CommandListener {

    //Creamos las variables necesarias
    public static SPPServidorMIDlet SPPs= null;
    public static Display display;

    private SPPServidor s =null;

    //Constructor
        public SPPServidorMIDlet() {
        SPPs = this;
    }

    //Implementamos el ciclo de vida del MIDlet
    public void startApp() {
        display = Display.getDisplay(this);
        s = new SPPServidor();
        s.inicializar();

        display.setCurrent(s);
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }

    //Este metodo se encarga de las tareas necesarias para salir del MIDlet
    public void salir(){
        SPPs.destroyApp(true);
        SPPs.notifyDestroyed();
        SPPs = null;
    }

    //Manejamos la accion del usuario
    public void commandAction(Command c, Displayable d) {
        if (d == s && c.getLabel().equals("Salir")) {
        //Salimos de la aplicacion
            try{
                s.fin = true;
```

```
                s.servidor.close();
            }
            catch(Exception e){}
        salir();
        }
    }
}
```

SPPServidor

```
package SPPBluetooth;

import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import javax.bluetooth.*;
import java.io.*;

public class SPPServidor extends List implements Runnable{

    //Objetos Bluetooth necesarios
    public LocalDevice dispositivoLocal;
    public DiscoveryAgent da;
    public boolean fin = false;
    public StreamConnectionNotifier servidor;


    //Constructor
    public SPPServidor(){
        super("Servidor SPP",List.EXCLUSIVE);

        addCommand(new Command("Salir",Command.EXIT,1));

        setCommandListener(SPPServidorMIDlet.SPPs);
    }

    //Este metodo se encarga de dar un aviso de alarma cuando se produce una excepcion
    public void mostrarAlarma(Exception e, Screen s, int tipo){
        Alert alerta;
        if(tipo == 0){
            alerta = new Alert("Excepcion:","Se ha producido la excepcion
"+e.getClass().getName(), null, AlertType.ERROR);
        }else{
            alerta = new Alert("Error:","No ha seleccionado un dispositivo ", null,
AlertType.ERROR);
        }
        alerta.setTimeout(Alert.FOREVER);
        SPPServidorMIDlet.display.setCurrent(alerta,s);
    }

    //Este metodo se va a encargar de inicializar el servidor
    public void inicializar(){
        try{
            dispositivoLocal = LocalDevice.getLocalDevice();
            dispositivoLocal.setDiscoverable(DiscoveryAgent.GIAC);

            //Lanzamos un hilo servidor (solo aceptara un cliente)
            Thread hilo = new Thread(this);
            hilo.start();
        }
        catch(BluetoothStateException be){
        System.out.println("Se ha producido un error al inicializar el hilo servidor");
        }
    }

    public void run(){
        //Le damos un nombre a nuestra aplicacion
        String nombre = "Ejemplo SPP";

        //Definimos un UUID unico para este servicio. Elegimos uno a nuestro gusto.
        UUID uuid = new UUID(0xABCD);
        servidor = null;//Similar a un socket servidor
        StreamConnection sc = null;//Similar a un socket cliente

        RemoteDevice rd = null;

        //Intentamos crear una conexion, usando SPP
```

```
        try{
            servidor =
(StreamConnectionNotifier)Connector.open("btspp://localhost:"+uuid.toString()+";name="+n
ombre);
            //Obtenemos el service record
            ServiceRecord rec = dispositivoLocal.getRecord(servidor);
            //Rellenamos el BluetoothProfileDescriptionList usando el SerialPort version
1
            DataElement e1 = new DataElement(DataElement.DATSEQ);
            DataElement e2 = new DataElement(DataElement.DATSEQ);
            e2.addElement(new DataElement(DataElement.UUID,new
UUID(0x1101)));//agregamos el puerto serie
            e2.addElement(new DataElement(DataElement.INT_8,1));//version 1
            e1.addElement(e2);
            //agregamos al service record el BluetoothProfileDescriptionList
            rec.setAttributeValue(0x0009,e1);
        }
        catch(Exception e){
            System.out.println("Se ha producido un error al lanzar el hilo servidor");
            mostrarAlarma(e, this,0);
            return;
        }

        while(!fin){
            try{
                append("Esperando mensaje:",null);
                //Aceptamos conexiones del cliente y obtenemos el objeto remoto
                sc = servidor.acceptAndOpen();
                rd = rd.getRemoteDevice(sc);
                //Obtenemos el input stream del objeto remoto
                DataInputStream in = sc.openDataInputStream();
                //Leemos el mensaje y lo mostramos por pantalla
                append(in.readUTF(),null);
                //Cerramos la conexion
                sc.close();
            }
            catch(Exception e){
                mostrarAlarma(e,this, 0);
                return;
            }
        }
    }
}
```

SPPClienteMidlet

```
package SPPBluetooth;


import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import javax.bluetooth.*;
import java.io.*;
import java.util.*;

public class SPPClienteMIDlet extends MIDlet implements CommandListener {

    //Creamos las variables necesarias
    public static SPPClienteMIDlet SPPc= null;
    public static Display display;

    private SPPCliente c =null;
    private Mensaje msg = null;

    //Objetos Bluetooth necesarios
    public LocalDevice dispositivoLocal;
    public DiscoveryAgent da;

    //Lista de dispositivos y servicios encontrados
    public static Vector dispositivos_encontrados = new Vector();
    public static Vector servicios_encontrados = new Vector();

    public static int dispositivo_seleccionado = -1;

    //Constructor
    public SPPClienteMIDlet(){
```

```
        SPPc = this;
    }

    //Implementamos el ciclo de vida del MIDlet
    public void startApp() {
        display = Display.getDisplay(this);

        c = new SPPCliente();
        msg = new Mensaje();

        //Mostramos la lista de dispositivos(vacia al principio)
        c.mostrarDispositivos();
        display.setCurrent(c);
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }

    //Este metodo se encarga de las tareas necesarias para salir del MIDlet
    public static void salir(){
        SPPc.destroyApp(true);
        SPPc.notifyDestroyed();
        SPPc = null;
    }

    //Este metodo se encarga de dar un aviso de alarma cuando se produce una excepcion
    public void mostrarAlarma(Exception e, Screen s, int tipo){
        Alert alerta=null;
        if(tipo == 0){
            alerta = new Alert("Excepcion","Se ha producido la excepcion
"+e.getClass().getName(), null, AlertType.ERROR);
        }
        else if(tipo==1){
            alerta = new Alert("Error","No ha seleccionado un dispositivo ", null,
AlertType.ERROR);
        }
        else if(tipo==2){
            alerta = new Alert("Informacion","El mensaje ha sido enviado ", null,
AlertType.INFO);
        }
        alerta.setTimeout(Alert.FOREVER);
        display.setCurrent(alerta,s);
    }

    //Manejamos la accion del usuario
    public void commandAction(Command co, Displayable d){
        if(d==c && co.getLabel().equals("Busqueda")){
            //Limpiamos la lista
            dispositivos_encontrados.removeAllElements();
            servicios_encontrados.removeAllElements();
            try{
                dispositivoLocal = LocalDevice.getLocalDevice();
                dispositivoLocal.setDiscoverable(DiscoveryAgent.GIAC);
                da = dispositivoLocal.getDiscoveryAgent();
                da.startInquiry(DiscoveryAgent.GIAC,new Listener());

                c.escribirMensaje("Por favor espere...");
            }
            catch(BluetoothStateException be){
                mostrarAlarma(be,c, 0);
            }
        }
        else if(d==c && co.getLabel().equals("Enviar")){
            dispositivo_seleccionado = c.getSelectedIndex();
            //Nos aseguramos de que el usuario selecciono un dispositivo
            if(dispositivo_seleccionado == -1 ){
                mostrarAlarma(null, c,1);
                return;
            }
            display.setCurrent(msg);
        }
        else if(d==c && co.getLabel().equals("Salir")){
            salir();
        }
```

```
        else if(d==msg && co.getLabel().equals("OK")){
            servicios_encontrados.removeAllElements();
            //Buscamos el servicio de puerto serie en el dispositivo seleccionado
            RemoteDevice dispositivo_remoto
=(RemoteDevice)dispositivos_encontrados.elementAt (dispositivo_seleccionado);
            try{
                //Buscamos en el puerto serie 0x1101
                da.searchServices(null,new UUID[]{new
UUID(0x1101)},dispositivo_remoto,new Listener());
            }
            catch(BluetoothStateException be){
                mostrarAlarma(be, c, 0);
            }
        }
    }

    //Este metodo se va a encargar de enviar un mensaje al primer ServiceRecord usando
el
    //Serial Port Profile
    public void enviarMensaje(String msg){
        ServiceRecord sr = (ServiceRecord)servicios_encontrados.elementAt(0);
        //Obtenemos la URL asociada a este servicio en el dispositivo remoto
        String URL = sr.getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOENCRYPT,false);
        try{
            //Obtenemos la conexio y el stream de este servicio
            StreamConnection con = (StreamConnection)Connector.open(URL);
            DataOutputStream out = con.openDataOutputStream();
            //Escribimos datos en el stream
            out.writeUTF(msg);
            out.flush();
            //Cerramos la conexion
            out.close();
            con.close();
            mostrarAlarma(null, c, 2);
        }
        catch(Exception e){
            mostrarAlarma(e, c, 0);
        }
    }

    //Implementamos el DiscoveryListener
    public class Listener implements DiscoveryListener{

        //Implementamos los metodos del interfaz DiscoveryListener

        public void deviceDiscovered(RemoteDevice dispositivoRemoto, DeviceClass clase){
            System.out.println("Se ha encontrado un dspositivo Bluetooth");

            dispositivos_encontrados.addElement(dispositivoRemoto);
        }

        public void inquiryCompleted(int completado){
            System.out.println("Se ha completado la busqueda de dispositivos");

            if(dispositivos_encontrados.size()==0){
                Alert alerta = new Alert("Problema","No se ha encontrado
dispositivos",null, AlertType.INFO);
                alerta.setTimeout(3000);
                c.escribirMensaje("Presione descubrir dispositivos");
                display.setCurrent(alerta,c);
            }
            else{
                c.mostrarDispositivos();
                display.setCurrent(c);
            }
        }

        public void servicesDiscovered(int transID, ServiceRecord[] servRecord){
            System.out.println("Se ha encontrado un servicio remoto");
            for(int i=0;i<servRecord.length;i++){
                ServiceRecord record = servRecord[i];
                //Se adiciona el servicio al vector de servicios encontrados
                servicios_encontrados.addElement(record);
            }
        }
        public void serviceSearchCompleted(int transID, int respCode){
            System.out.println("Terminada la busqueda de servicios");
```

```
            //Si encontramos un servicio, lo usamos para mandar el mensaje(todos los
            //servicios que hemos buscado son de puerto serie)
            if(servicios_encontrados.size()>0){
                enviarMensaje(msg.getString());
            }
            else{
                //Si no encontramos ningun servicio de puerto serie
                c.mostrarDispositivos();
                display.setCurrent(c);
            }
        }
    }
}
```

SPPCliente

```
package SPPBluetooth;


import javax.microedition.lcdui.*;
import javax.bluetooth.*;


public class SPPCliente extends List{

    public SPPCliente(){
        super("Cliente SPP",List.EXCLUSIVE);
        addCommand(new Command("Busqueda",Command.SCREEN, 1));
        addCommand(new Command("Enviar",Command.SCREEN, 1));
        addCommand(new Command("Salir",Command.EXIT, 1));

        this.setCommandListener(SPPClienteMIDlet.SPPc);
    }

        //Este metodo se encarga de limpiar la pantalla y de mostrar un mensaje
    public void escribirMensaje(String str){
            append(str,null);
    }

    //Este metodo muestra los "friendly names" de los dispositivos remotos
    public void mostrarDispositivos(){
        this.deleteAll();
            if(SPPClienteMIDlet.dispositivos_encontrados.size()>0){
                for(int i=0;i<SPPClienteMIDlet.dispositivos_encontrados.size();i++)
                {
                    try{
                        RemoteDevice dispositivoRemoto = (RemoteDevice)
SPPClienteMIDlet.dispositivos_encontrados.elementAt(i);
                        append(dispositivoRemoto.getFriendlyName(false),null);
                    }catch(Exception e){
                        System.out.println("Se ha producido una excepcion");
                    }
                }
            }
        else append("Pulse Busqueda",null);
    }
}
```

Mensaje

```
package SPPBluetooth;

import javax.microedition.lcdui.*;

public class Mensaje extends TextBox{

    public Mensaje(){
        super("Introducir mensaje","Hola Mundo",50,TextField.ANY);

        addCommand(new Command("OK",Command.SCREEN, 1));
        this.setCommandListener(SPPClienteMIDlet.SPPc);
    }
}
```

3.5.7 Cliente y servidor.

En este apartado se explicarán algunos detalles de la implementación, tanto del cliente J2ME como del servidor J2SE. Ya se ha visto la teoría acerca de la inicialización de la pila, búsqueda de dispositivos y servicios, establecimiento de la conexión y comunicación, y ahora se verá en la práctica que decisiones de diseño e implementación se han tomado en el diseño final.

3.5.7.1 Cliente.

En cualquier aplicación basada en Bluetooth, una vez inicializada la pila, se procedería a buscar los dispositivos cercanos para luego buscar los servicios que ofrecen y que nos interesan. Para ello lo típico es mostrar en pantalla los dispositivos encontrados para luego seleccionar uno de ellos y buscar sus servicios. En este proyecto todo eso no es necesario, pues nos conectaremos al dispositivo (en este caso un PC servidor) que indica la tarjeta NFC y por ello no es necesario mostrar ninguna búsqueda, ni de dispositivos ni de servicios.

Es por ello, que nuestra aplicación nada mas arrancar, mostrará la pantalla de lectura de la tarjeta NFC, mientras en un segundo plano transparente al usuario se estará llevando a cabo una búsqueda de dispositivos cercanos.

Una vez leída la tarjeta ya tendremos la dirección bluetooh del PC servidor, y solo hará falta ver si éste esta entre los dispositivos encontrados en la búsqueda. La siguiente pantalla que mostrará la aplicación será la de selección del archivo de envío (ver apartado del file browsing). Se puede recorrer el sistema local de ficheros para seleccionar el archivo que se desea enviar al servidor para su proyección. Un vez seleccionada, la aplicación comprobará que en la búsqueda de dispositivos se encuentra el dispositivo de la tarjeta NFC. En caso afirmativo se llevará a cabo una búsqueda del servicio para envió de archivos. Podríamos pensar que este paso es innecesario, pues si ya conocemos el dispositivo servidor sabemos que ofrece ese servicio, pero a pesar de ello la búsqueda de servicios hay que realizarla obligatoriamente pues hay que implementar este método en nuestro Midlet.

Si todos los pasos anteriores se han llevado a cabo correctamente, solo falta obtener la URL del servidor para abrir la conexión y enviar el archivo. Una vez abierta la conexión para el envío de los archivos obtendremos los flujos de entrada y salida correspondientes. El protocolo que se seguirá consiste en enviar primero el nombre y extensión del archivo (para que así el servidor sepa recomponerlo correctamente) y luego enviarle los datos mediante un array de bytes.

3.5.7.2 Servidor.

La parte del PC servidor se implementó primero en J2ME, para así poder realizar la simulación en el propio ordenador sin necesidad de disponer de un terminal Nokia 6131 que hiciera de cliente y un pendrive bluetooth para el servidor.

Una vez que se comprobó el funcionamiento correcto de ambas partes, y que la transferencia de ficheros se hacia bien, se procedió a pasar la aplicación J2ME del servidor a J2SE. El principal problema que conlleva esto es la falta de librerías de Bluetooth en J2SE, por lo que se tuvo que recurrir a Bluecove.

Bluecove es una implementación libre del API JSR-82 en J2SE. Gracias a Bluecove tan solo haciendo unos pequeños cambios en el código que teníamos programado para la aplicación servidora en J2ME, se consiguió que el servidor

funcionase en el ordenador y pudiese realizar todas sus funciones usando un pendrive usb de bluetooth que se le incorporó.

La librería Bluecove no es mas que un conjunto de clases del API JSR-82 contenidas en un archivo .jar, el cual habrá que añadir al classpath en el momento de la compilación y la ejecución para que funcione.

En la siguiente figura se muestra podemos ver donde se situaría la librería en la torre de protocolos de Bluetooth:

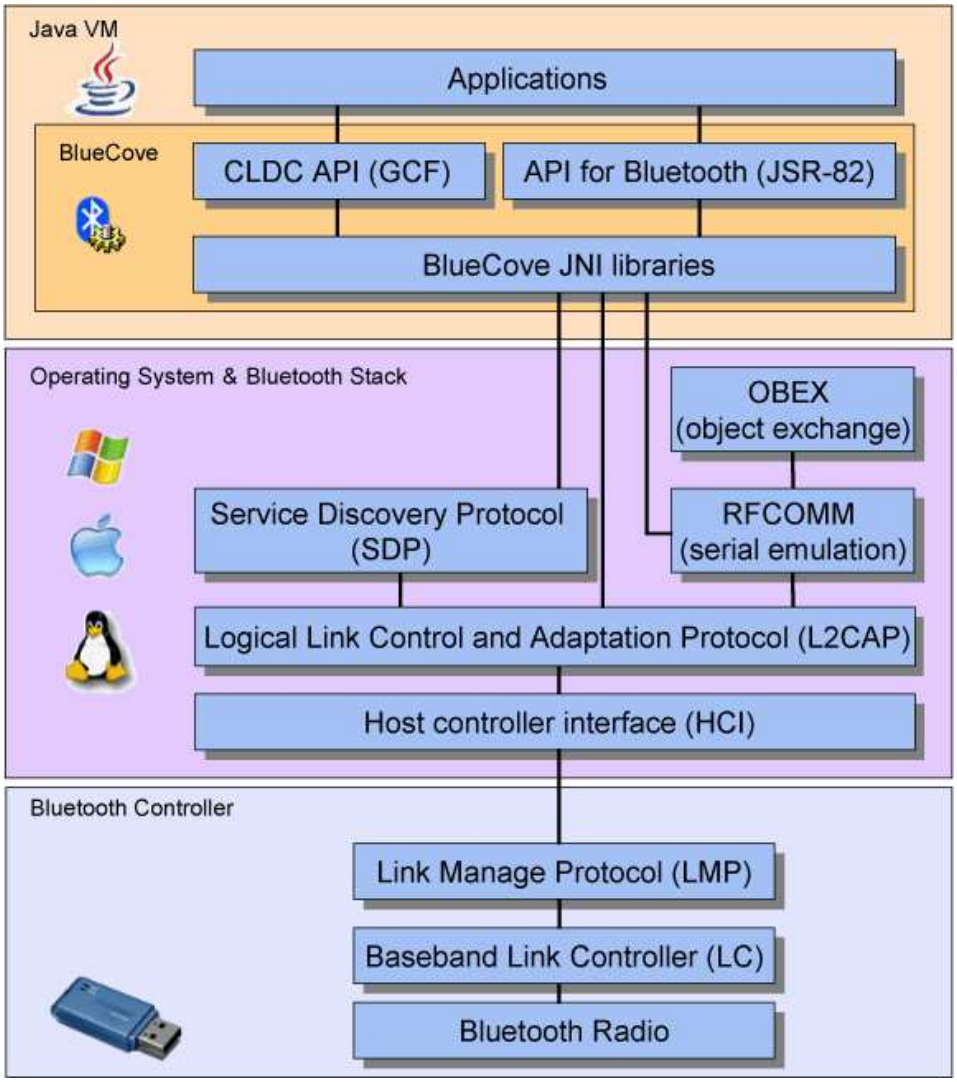

Ilustración 30. Librería Bluecove en la torre de protocolos Bluetooth.

3.6. Batería de pruebas.

En este apartado vamos a ver la batería de pruebas que se le ha ido pasando al proyecto, tanto por funcionalidades principales como en bloque, para poder comprobar su correcto funcionamiento.

Las pruebas de software son una parte esencial en el desarrollo de una aplicación, ya sea a nivel personal como profesional. Éstas se integran en el ciclo de vida del desarrollo del software e incluye todos aquellos procesos que permiten verificar el funcionamiento de una aplicación así como revelar la calidad de ésta.

Las pruebas de software son un proceso usado para identificar posibles fallos de implementación, calidad, o usabilidad de un software. Básicamente es una fase que consistente en probar las aplicaciones construidas. "El testing puede probar la presencia de errores pero no la ausencia de ellos", E. W. Dijkstra.

3.6.1 Escritura y lectura de tarjetas.

En esta parte se probó el correcto funcionamiento a la hora de escribir en las tarjetas NFC para su posterior lectura. Para ello se implemento la aplicación complementaria que se llama TagWriter. Las características de esta aplicación ya se vieron en el apartado 3.3.1, por lo que no entraremos en más detalle, sino que veremos como se probó tanto en el simulador como en el dispositivo final.

En primer lugar, antes de pasar la aplicación al terminal cliente, se hicieron algunas pruebas como el simulador de Nokia para comprobar que efectivamente la aplicación escribía la información deseada en las tarjetas. Para ello arrancamos la aplicación en el simulador escogiendo el archivo TagWriter.jad. La vista del simulador es la siguiente:

Ilustración 31. Vista de la aplicación TagWriter en el simulador.

Como podemos ver, a la izquierda tenemos la vista de la aplicación, y a la derecha, dividido en dos partes, las diferentes tarjetas emuladas. Para escribir en una tarjeta bastaría con acercar el móvil a la tarjeta en la vida real, pero en simulación ésta acción se hace mediante un doble click en la tarjeta que deseamos escribir.

El aplicación nos muestra que se ha escrito correctamente la tarjeta la hacer doble clic, así que guardamos los datos en la tarjeta eligiendo la opción “Save data”. Esto es necesario para que el simulador guarde esta información el la tarjeta para las siguientes emulaciones que se hagan con éstas.

Ilustración 32. Vista de la aplicación TagReader en el simulador.

En principio vemos que todo funciona a la perfección, pero debemos comprobar si realmente se ha guardado la información en la tarjeta y si lo ha hecho correctamente y sin errores. Para ello utilizamos la aplicación TagReader, que implementamos para esto mismo.

Con la aplicación TagReader lanzada en el simulador, se procedió a la lectura de la tarjeta previamente escrita. Al igual que a la hora de escribir la acción de tocar una tarjeta se hacia mediante un doble click, en el caso de la lectura es igual. Haciendo esto se comprobó que efectivamente se leían los mismos datos que se habían escrito en el paso anterior.

El siguiente paso, para cerciorarnos de que todo funcionaba correctamente, fue cargar las dos aplicaciones en el móvil y hacer lo mismo. Tanto la escritura de la tarjeta, como la lectura funcionaron a la primera, cosa que me sorprendió. Un detalle a tener en cuenta es que en ambos procesos, lectura y escritura, no se debe alejar el teléfono de la tarjeta pues puede llevar a una mala operación y posterior funcionamiento erróneo. De éste modo, dejamos el teléfono cerca de la tarjeta hasta que recibimos confirmación por parte de la aplicación.

Si tenemos dos tarjetas juntas, la aplicación leerá y escribirá en las dos a la vez, por lo que es necesario que tengamos siempre separada la tarjeta objetivo. En nuestro caso, solo tendremos una tarjeta en el escenario pro tanto éste caso nunca se dará, pero por si acaso se quiso ver que ocurría.

3.6.2 Descubrimiento de dispositivos.

Con estas pruebas se quiso determinar, antes de integrar esta funcionalidad de la aplicación, el correcto funcionamiento a la hora de descubrir los dispositivos cercanos por la aplicación cliente.

Para ello, a diferencia de las pruebas que se hicieron para la parte de NFC, se utilizó el simulador del Wireless Toolkit porque el simulador de Nokia dio algunos problemas y aun no sabemos porqué fue.

Se programó una pequeña aplicación cuyo objetivo era buscar los dispositivos que se encontraban en el alcance y buscar un determinado servicio en uno de esos dispositivos. Para ello se programó un pequeño servidor que ofrecía un servicio y arrancamos éste en otro simulador. Wireless Toolkit permite que una aplicación bluetooth en un simulador (por ejemplo un cliente) detecte a otra (un servidor) que esta funcionando en otro simulador. En la siguiente figura podemos ver la vista de la aplicación y como encontraba correctamente al servidor:

Ilustración 33. Descubrimiento de dispositivos en el emulador.

Para la búsqueda de dispositivos no hace mas falta que pulsar en buscar, y la aplicación a medida que encontraba dispositivos en su alcance los añadía la vista. Si pulsamos en seleccionar se procedía a la búsqueda de un servicio en concreto, que no era más que el que habíamos programado para que el servidor ofreciera.

Viendo que funcionaba en el simulador, se pasó la aplicación al móvil para ver si también funcionaba en un entorno real. A un PC le conectamos un pendrive bluetooh y con mi teléfono personal active el bluetooh. La aplicación en el móvil era capaz de detectar a ambos dispositivos.

Una vez comprobado esto, se integró en la aplicación final, pero de una manera distinta que ya se vió en el apartado de la implementación de bluetooth. La aplicación cliente no mostrará en ningún momento una pantalla de búsqueda pues al leer la tarjeta ya sabemos a que dispositivo hay que enviarle los datos. La búsqueda se lleva a cabo pero en un segundo plano transparente para el usuario, para si comprobar que el dispositivo que indica la tarjeta esta efectivamente en el alcance.

3.6.3 Conexiones en el servidor.

3.6.4 Envío de datos al servidor.

Después de comprobar que la aplicación cliente era capaz de conectarse al servidor correctamente, se pasó a probar el envío de datos. En primer lugar solo se enviaban imágenes, pues no supe como reconstruir otro tipo de archivos en el destino (servidor). El problema era que los bytes se enviaban correctamente, pero al guardarlos en destino algo fallaba y al abrir el archivo no se podía ver.

Se fueron enviando archivos de imagen cada vez más grandes, hasta que se vio que había una limitación de tamaño en las transferencias, de tal modo que solo se podían enviar archivos de hasta 1 MB. Si se trata de enviar archivos de mayor tamaño salta una excepción indicándoos un fallo en la capacidad de la memoria, es decir, que la aplicación no era capaz de guardar un array de mayor tamaño para luego pasarlos al archivo y reconstruirlo.

En principio, esto no nos afecta mucho, pues uno de los objetivos del proyecto era la proyección de imágenes y pequeños documentos, así que la limitación no nos afecta para el escenario que estamos tratando. De todas maneras esto podría solucionarse dividiendo el archivo en pequeños paquetes e irlos enviando uno tras otro siguiendo un protocolo, y que así el servidor fuera escribiendo los datos en el destino poco a poco, no afectando de ésta manera esa limitación de capacidad. La implementación de este protocolo de envío podría ser una interesante futura línea de trabajo para mejorar el proyecto.

Tras un tiempo, leyendo mucha documentación acerca del manejo de archivos en j2me, se encontró una solución para poder enviar cualquier tipo de archivo y que éste fuera reconstruido en origen correctamente y abierto con una determinada aplicación por defecto. El uso de la clase Desktop, y mas en concreto el método open (File f), permite abrir el archivo especificado con la aplicación por defecto establecida en el ordenador.

Solucionado esto, se empezaron a enviar diferentes archivos y todos los abría perfectamente el servidor independientemente de su extensión. De este modo se podían enviar documentos de Word (.doc), presentaciones en Powerpoint (.ppt), documentos pdf, imágenes en diferentes formatos, videos, sonido, etc…

4. Conclusiones y trabajos futuros.

4.1 Conclusiones.

En este proyecto fin de carrera se ha implementado una aplicación para dispositivos móviles que permite la proyección de contenidos multimedia de manera distribuida. Se han estudiado dos tecnologías inalámbricas de comunicaciones, como son Bluetooth y Near Field Communication, que resultan muy adecuadas e interesantes para aplicaciones dirigidas a funcionar en dispositivos móviles debido al gran abanico de posibilidades que ofrecen. Se ha estudiado también la plataforma y entorno de desarrollo J2ME, muy útil a la hora de diseñar y desarrollar aplicaciones para móviles.

Para probar y asentar todos estos conocimientos se ha desarrollado éstas aplicación y nos ha permitido conocer el grado de usabilidad que tienen todas estas tecnologías hoy en día.

Por ultimo se diseñaron una serie de pruebas que sirvieron tanto para probar el correcto funcionamiento de la aplicación, clientes y servidor, como para depurar algunos aspectos de ésta misma. Esta batería de pruebas no se ha pasado una vez integradas todas las funcionalidades en la aplicación final, si no que se fueron pasando para ir probando funcionalidades independientes y luego el funcionamiento de la aplicación en conjunto.

Las tecnologías Bluetooth y NFC, son totalmente costeables a escala empresarial, la integración con dispositivos móviles ofrece un ahorro de tiempo y costes en algunos aspectos. Esta integración puede hacer incrementar la productividad y darle ventajas competitivas a aquellas compañías que la adopten. Sin embargo hay escenarios donde aplicar esta tecnología presenta una serie de dificultades, ya sea debido a costes, seguridad, almacenamiento de datos, etc. De todas maneras, la integración en sistemas de la información existentes en las compañías les posibilita a tener un control potencial sobre las inversiones en equipos.

Como se ha mostrado en el apartado del estado del arte, ambas tecnologías inalámbricas, integradas en el teléfono móvil ofrecen un gran número de posibilidades y casos de uso, sin resultar intrusivas para el ser humano como usuario de la misma. Además, tampoco se conciben como unas tecnologías que eliminen a otras, ya que pueden convivir perfectamente con la tecnología ya existentes, haciendo que los servicios que ya se ofrecen se puedan extender, facilitando así el día a día de los usuarios en sus actividades.

La integración de NFC y J2ME es una buena combinación que puede incrementar la productividad y darle ventajas competitivas a aquellas empresas que la adopten. Muchos fabricantes de teléfonos móviles están descubriendo el gran potencial de la unión de la tecnología NFC con dispositivos móviles, por ello algunos como Samsung, quieren que para el 2010 el 50% de los teléfonos en el mercado tengan estas capacidades.

En cuanto a la herramienta en si, se ha conseguido desarrollar una aplicación basada en dos tecnologías que se están en pleno auge en la actualidad. La sencillez y su facilidad de uso, hace que ésta herramienta pueda ser usada sin necesidad de unos conocimientos muy altos de ambas tecnologías. Ya que esta herramienta esta dirigida a todo tipo de usuarios (desde un alto ejecutivo de una empresa, hasta un investigador de i+d, pasando por programadores, secretarias o administrativos) se ha

buscado que ésta resulte muy intuitiva, con menús claros y sencillos. Además, como ya se vio en los objetivos del proyecto se ha buscado que la aplicación fuera rápida (la transferencia de ficheros mediante bluetooth resulta….), escalable (en ningún momento existe un número limitado de clientes que puedan conectarse al servidor), dinámica (clientes pueden dejar de participar sin ningún tipo de aviso, así como la participación de nuevos clientes), etc.

4.2 Futuras líneas de trabajo.

Varias son las posibles líneas de trabajo en el futuro, de tal manera que otro alumno o programador pueda continuar con este proyecto, mejorarlo o añadirle mas funcionalidades. Las principales líneas de trabajo que se me ocurren, las enumero a continuación:

- Aumento en la capacidad de transferencia de archivos. Como se pudo ver en el apartado de pruebas existía una limitación a la hora de transferir archivos desde el teléfono móvil al servidor, que era de aproximadamente 1 MB. EN principio esta limitación no nos afecta mucho pues el tipo de archivos que íbamos a enviar no llegaban a ese tamaño, de todas maneras sería interesante ampliarlo. Para ello bastaría con diseñar un protocolo entre cliente y servidor de tal manera que el cliente dividiese el archivo en varios paquetes, le informase al servidor de cuantos paquetes consta la transferencia, e ir enviándoselos uno a uno esperando una confirmación. El servidor a medida que fuera recibiendo los paquetes los podría ir juntando y reconstruyendo el archivo original.

- Seguridad: en algunos escenarios de uso podría resultar interesante implementar algún tipo de autenticación por parte de los clientes en el servidor, de tal manera que el servidor estuviera configurado para permitir la conexión a una serie de dispositivos “conocidos”. Todo esto sería una interesante línea de trabajo futura siempre y cuando no conllevase un aumento

- Registro de archivos: podría ser interesante integrar una nueva funcionalidad que consistiese en que el servidor llevase algún tipo de control sobre los archivos que va recibiendo, de tal manera que tuviese un archivo de registro que contuviese el listado de los archivos que se han enviado y quien los ha enviado.

- Vista previa: otra funcionalidad que se podría implementar sería la de poder ver el archivo en el teléfono móvil antes de enviarlo al servidor. Dado el caso de que tengamos muchos archivos en la memoria del teléfono resultaría muy interesante poder hacer una vista previa para poder comprobar que efectivamente ese es el archivo que deseamos enviar.

5. Presupuesto del proyecto y desarrollo temporal.

Para el cálculo del presupuesto total del proyecto fin de carrera se va a especificar el número de horas dedicadas a cada una de las tareas que componen las distintas fases en que se puede desglosar:

1) Análisis: 75 horas.
   - 1.1 Estudio de requisitos y formulación de objetivos: 10 horas.
   - 1.2 Recopilación de material de estudio: 15 horas.
   - 1.3 Estudio de tecnologías para la implementación del sistema: 20 horas.
   - 1.4 Familiarización con el entorno de trabajo y realización de demostraciones previas de las tecnologías elegidas: 30 horas.

2) Diseño: 100 horas.
   - 2.1 Definición de la arquitectura: 20 horas.
   - 2.2 Diseño de la aplicación cliente: 50 horas.
   - 2.3 Definición de la aplicación servidora: 30 horas.

3) Implementación: 215 horas.
   - 3.1 Planificación de la fase de desarrollo: 15 horas.
   - 3.2 Implementación NFC: 60 horas.
   - 3.3 Implementación file-browsing: 50 horas.
   - 3.4 Implementación bluetooth cliente: 60 horas.
   - 3.5 Implementación bluetooth servidor: 30 horas.

4) Pruebas de usuario: 40 horas.
   - 4.1 Planificación y desarrollo de batería de pruebas: 10 horas.
   - 4.2 Ejecución de programas de pruebas manuales: 20 horas.
   - 4.3 Recogida y análisis de resultados: 5 horas.
   - 4.4 Corrección de errores: 5 horas.

Por tanto, la suma total de horas dedicadas a la realización del proyecto final de carrera asciende a 430 horas. Para el cálculo relacionado con los costes de personal se va a aplicar la tarifa que propone el Colegio Oficial de Ingenieros Técnicos de Telecomunicación. El valor de la tarifa de Honorarios para el año 2009 no se ha encontrado así que se utilizará la de 2006, de manera orientativa. Es importante destacar que el proyecto fin de carrera ha sido realizado por una persona que todavía no está licenciada por lo que a efectos del presupuesto se considerará el valor de la hora trabajada como el 70% del valor de la hora normal para un ingeniero ya licenciado.

Coste de personal			
Concepto	Horas	Tarifa	Total
Ing. Técnico no licenciado	430 horas	42,11 €	18107,3 €

Hay que tener en cuenta además, no solo el coste de personal anteriormente calculado, sino también el coste de material que se describe en la siguiente tabla:

Coste de material				
Concepto	Precio unitario	Coeficiente de amortización	Unidades	Importe
Ordenador portátil	700 €	1/3	1	233,33 €
Teléfono Nokia 6131 NFC	200 €	1/3	1	66,66 €
Pendrive Bluetooth	20 €	1/3	1	6,66 €
Impresión y bibliografía	400	1/1	1	400 €
			Subtotal	706,65 €

Se ha supuesto que el periodo de vida media de un ordenador personal y del software utilizado para la realización del proyecto es de 3 años y que el tiempo total de uso ha sido de 1 año.

El coste total del proyecto quedaría reflejado en la siguiente tabla:

Coste total	
Concepto	Importe
Coste de personal	18107,3 €
Coste del material	706,65 €
Base imponible	18813,95 €
I.V.A. (16%)	3010,23 €
Total	**21824,18 €**

Por tanto el coste total del proyecto fin de carrera asciende a VEINTIÚN MIL OCHOCIENTOS VEINTICUATRO CON DIECIOCHO CENTIMOS (21824,18 €).

Apéndice A: Manual de instalación.

A.1 Introducción

Se describirá como el administrador local deberá de instalar y mantener el software necesario para el correcto funcionamiento de la herramienta. También se explicará qué herramientas software fueron necesarias instalar y configurar para el desarrollo de esta aplicación así como el modo en el que estas fueron instaladas y configuradas.

A.2 Java JDK 6 Update 14.

Este es el entorno de desarrollo para Java proporcionado por Sun Microsystems y que utilizaremos para compilar nuestras clases Java. También nos proporciona la herramienta **JavaDoc**, que nos permitirá documentar las clases.

El **JDK** lo podremos bajar desde:
[http://java.sun.com/javase/downloads/index.jsp]]

Una vez descargado el archivo ejecutable **jdk-6u14-windows-i586.exe** se procede a su instalación. Una vez instalado, tendremos que crear una serie de variables de entorno para acabar la instalación y poder ponernos a trabajar con el entorno de desarrollo.

Para la creación de las variables deberemos de acudir (en Windows XP) a: **Mi PC > Propiedades > Opciones Avanzadas > Variables de Entorno.** Una vez aquí deberemos de crear las siguientes dos variables de entorno:
CLASSPATH = <PATH JDK>;

Así indicaremos al compilador Java donde ha de ir para encontrar los directorios que contienen las clases o librerías de Java (el API de Java). En un futuro tendremos que añadir a la variable **CLASSPATH** aquellos nuevos directorios en los que queremos que el compilador sea accesible a nuevas clases de usuario.
PATH = <PATH JDK>\bin;

El desarrollo y ejecución de aplicaciones en Java exige que las herramientas para compilar (**javac.exe**) y ejecutar (**java.exe**) se encuentren accesibles. El ordenador, desde una ventana de comandos de MS-DOS, sólo es capaz de ejecutar programas que se encuentre en los directorios indicados en la variable **PATH** del ordenador. Si se desea compilar o ejecutar código en Java en estos casos el directorio donde se encuentran estos programas (java.exe y javac.exe) deberán encontrarse en el **PATH**.

A.3 Wireless Toolkit 2.5.2.

Wireless Toolkit es una herramienta para el desarrollo de aplicaciones basadas en J2ME y diseñadas para funcionar en teléfonos móviles, PDAs y otros dispositivos.

Se puede descargar desde la siguiente ubicación:
[http://java.sun.com/products/sjwtoolkit/download.html]

A.4 SDK Nokia 6131 NFC.

Este SDK complementario permite al programador crear aplicaciones para el teléfono móvil Nokia 6131 NFC. Incluye librerías propias, como por ejemplo, la necesaria para la funcionalidad de NFC, y un simulador del teléfono. También permite integrar el simulador en Wireless Toolkit instalando el SDK en su directorio.

Este SDK permite a los desarrolladores crear y emular aplicaciones Java ™ (MIDlets) para el teléfono móvil Nokia 6131 NFC. El SDK incluye API JSR-257, que permite el uso de la funcionalidad Near Field Communication (NFC). Además incluye varias extensiones de las tecnologías de la etiqueta, de igual a igual las conexiones y la configuración de la marca. El Nokia 6131 NFC SDK contiene un simulador del teléfono Nokia 6131 NFC, varias APIs de Java, algunos ejemplos de apliación (MIDlets) y documentación.

Es posible también la integración de este SDK en diferentes entornos como Wireless Toolkit o NetBeans, de tal manera que se puedan crear y compilar MIDlets, y ser probados en el emulador, directamente desde el entorno de desarrollo. El emulador se puede utilizar también como un programa independiente de la línea de comandos.

SDK Nokia 6131 NFC puede descargarse desde:
[http://sw.nokia.com/id/998af293-9ce7-4e8b-a7ab-1d63ad399a0c/Nokia_6131_NFC_SDK_1_1.zip]

Para integrarlo en Wireless Toolkit (como asi se ha hecho en este proyecto), basa con copiar la carpeta de instalación en:
<DIRECTORIO INSTALACIÓN WirelessToolkit>\ wtklib\devices

A.5 Librería Bluecove.

Bluecove es una implementación libre del API JSR-82 en J2SE. Gracias a Bluecove haciendo unos pequeños cambios en el código fuente es posible portar una aplicación Bluetooth J2ME a J2SE.

La librería Bluecove no es mas que un conjunto de clases del API JSR-82 contenidas en un archivo .jar, el cual habrá que añadir al classpath en el momento de la compilación y la ejecución para que funcione.

Bluecove se puede descargar desde la página del proyecto:
[http://www.bluecove.org/]

A.6 Nokia PC Suite.

Esta aplicación nos permitirá instalar aplicaciones en el teléfono móvil, así como transferir archivos y otras funcionalidades.

Este software se puede descargar en la siguiente dirección:
[http://www.nokia.es/soporte/software/pcsuite/descarga]

Normalmente al abrir el programa se detectará nuestro teléfono móvil y permitirá configurarlo. De no ser así, podemos abrir el asistente de conexión y configurar el acceso al teléfono. Gracias a su intuitivo menú no resultara difícil utilizar cualquiera de sus funcionalidades.

A.7 Editor del programador.

Existen numerosas posibilidades para elegir un programa editor de textos para programar Judit, JBuilder, Kedit, Xemacs, etc. En nuestro caso, se ha utilizado EditPlus por ser un excelente editor de textos que cumple con todas las necesidades de edición del programador. LA versión de prueba nos ofrece una licencia para 30 días, tras la cual es posible seguir utilizando el editor sin necesidad de comprar una licencia.

Apéndice B: Manual de usuario.

Este manual describe los pasos para la utilización de la aplicación en un escenario de proyección distribuida. Antes de proceder a utilizar la herramienta es necesaria la instalación previa de software adicional:

Servidor:
- Java JDK 6 Update 14.
- Librería Bluecove.

Cliente:
- Nokia PC Suite

En el manual de instalación podemos ver como instalar el software además que ubicación obtenerlo. De todas maneras, todo este software se encuentra contenido en el cd del proyecto.

Para arrancar el servidor en el PC basta con seguir los siguientes pasos:

1) Copiar la clase Servidor.java al directorio donde instalaremos el servidor, por ejemplo C:\Server\Servidor.java.
2) En este mismo directorio, guardamos la librería bluecove-2.1.0.jar.
3) Lanzamos el interfaz de línea de comandos: Inicio - Ejecutar – “cmd”
4) Compilamos, escribiendo en el *prompt:*

```
javac Servidor.java –cp .;bluecove-2.1.0.jar
```

5) Ejecutamos el servidor:

```
java Servidor nºtransacciones –cp .;bluecove-2.1.0.jar
```

Para arrancar la aplicación cliente en el teléfono móvil:
1) Abrimos el programa Nokia PC Suite:

2) Hacemos clic en el icono de “File Manager” (también File – File Manager). Se abrirá una ventana nueva similar al explorador de Windows. Buscamos el archivo Browser.jar y hacemos doble-click sobre él. Una ventana de confirmación nos pregunta si deseamos instalar la aplicación en el teléfono, y aceptamos.

3) Una vez instalada la aplicación, vamos al teléfono: Menu – Apps. Buscamos nuestra aplicación y en opciones damos a mover y la llevamos a la carpeta “Secure Applications”. Una vez allí, editamos los permisos de la aplicación en Options – Application access.

Communication – Network access – Ask first time only
Messaging – Ask every time
Connectivity –Always allowed
Data access – Mutimedia recording – Ask first time only
Read user data – Ask every time
Add and edit data – Ask every time
Auto-start – Not allowed

4) Abrimos la aplicación, pulsando en la opción open.

Llegados a este punto ya tenemos lanzado el servidor en el PC y la aplicación en el cliente. Basta con seguir los pasos que nos indica la aplicación para utilizar correctamente la herramienta de proyección de contenidos.

Bibliografía y referencias.

[1] **Margaret L. McLaughlin, Joao P. Hespanha y Gaurav S. Sukhatme.** "Touch in Virtual Environments: Haptics and the Design of Interactive Systems". Pearson Education. Diciembre 2001

[2] **Nathan J. Muller.** "Tecnología Bluetooth". McGraw-Hill. 2002.

[3] **Bala Kumar, Paul J. Kline, Timothy J. Thompson.** "Bluetooth Application Programming With the Java APIS". Morgan Kaufmann. 2004.

[4] **Lozano Ortega, Miguel Angel.** "Programación de dispositivos móviles en J2ME". Universidad de Alicante, Servicio de publicaciones. 2005.

[5] **Wikipedia**
[http://es.wikipedia.org/wiki/Near_Field_Communication]
[http://es.wikipedia.org/wiki/RFID]

[6] **Near Field Communication Research Lab Hagenberg**
[http://www.NFC-research.at/]

[7] **Java APIs for Bluetooth Wireless Technology (JSR 82) Specification**
Motorola. Wireless Software, Applications & Services

[8] **The Java APIs for Bluetooth Wireless Technology**
[http://developers.sun.com/mobility/midp/articles/bluetooth2/]

[9] **Bluetooth Technology Overview Version 1.0**
[http://www.forum.nokia.com]

[10] **Nokia 6131 NFC**
[http://europe.nokia.com/A4307094]

[11] **Bluetooth.com**
[http://www.bluetooth.com]

[12] **Bluetooth.org**
[http://www.bluetooth.org]

[13] **SIG Group**
[http://www.bluetooth.com/Bluetooth/SIG/]

[14] **JavaBluetooth.com**
[http://www.javabluetooth.com/]

[15] **"Getting Started with Java™ and Bluetooth"**
[http://today.java.net/pub/a/today/2004/07/27/bluetooth.html]

[16] **Pedro Daniel Borches Juzgado. "**Java 2 Micro Edition: Soporte Bluetooth**"**

[17] **Sun Java Wireless Toolkit**
[http://java.sun.com/products/sjwtoolkit/download.html]

[18] **SDK for Nokia 6131 NFC**
[http://www.forum.nokia.com/info/sw.nokia.com/id/ef4e1bc9-d220-400c-a41d-b3d56349e984/Nokia_6131_NFC_SDK.html]

[19] **Innovision Research & Technology**
[http://www.innovision-group.com/products.php]

[20] **TodoSymbian**
[http://www.todosymbian.com/]

[21] **Bluecove.**
[http://www.bluecove.org/]

[22] **NFC Forum**
[http://www.NFC-forum.org/]

[23] **NFC Forum. White Paper.** "Near Field Communication and the NFC Forum: The Keys to Truly Interoperable Communications". 2007
[http://www.NFC-forum.org/resources/white_papers/]

[24] **Nokia NFC**
[http://www.nokia.com/NFC]

[25**] Infrared Data Association**, *IrDA specifications.* 1993-2005.
[http://www.irda.org]

[26] **Mikhalenko, Peter V.** "Developing Wireless Bluetooth Applications in J2ME".
[http://java.sys-con.com/read/47688.htm]

[27] **Agustín Froufe Quintas y Patricia Jorge Cárdenas,** "Java 2 Micro Edition (J2ME). Manual de usuario y tutorial". Editorial Ra-Ma, edición 2004.